Issue 21 | Apr 2009

Blender learning made easy

blender art MAGAZINE

Look What I Can Do!

COVERART Breathe by Reynante

blender art MAGAZINE

EDITOR
Gaurav Nawani gaurav@blenderart.org

MANAGING EDITOR
Sandra Gilbert sandra@blenderart.org

WEBSITE
Nam Pham nam@blenderart.org

DESIGNER
Gaurav, Sandra, Alex

PROOFERS
Kevin Braun
Phillip Ryals
Bruce Westfall
Joshua Leung
Lynda Schemansky
Eric Pranausk
Noah Summers
Joshua Scotton
Mark Warren
Wade Bick
Patrick O'Donnell
Brian C. Treacy
Scott Hill
Henriel Veldtmann

WRITERS
Byron R. Kindig
Valentin Spirik
Claas Kuhnen
Benjamin Schram
Yap Chun Fei
Robert Burke
Sozap
Giancarlo Ng
Paul Fitzpatrick
Ken Beyer
SciVis IFC - CNR
Jeroen Bakker
Troy James Sobotka

COVER ART
Breathe - By Reynante
reynantem@gmail.com

www.blenderart.org

CONTENTS

Sandra Gilbert
Managing Editor

So if you have been feeling stuck in a creative rut or would just like to explore a new idea, just sit back and learn what others have been up to..

Over the last four years, we have explored and focused on various tips, techniques, tools and features of Blender.

And while we have covered various projects and artists over the years, the focus has tended toward "how" they did things. The "what" and "why" of their projects, assumed somewhat of a secondary nature.

When we picked the theme for this issue, I thought it would be a nice change of pace to see what the community at large is up to.

And we discovered that they are busy with a dizzying array of fun and interesting projects, that they were all too happy to share with the rest of us.

So if you have been feeling stuck in a creative rut or would just like to explore a new idea, just sit back and learn what others have been up to. Their creative uses for Blender are bound to spark a creative idea or two.

Have Fun!

sandra@blenderart.org

I normally do the greater majority of any project Planning in my head. Little did I know that this would be just the beginning of my "What not to do" lessons. ...

Introduction

Sometimes the most valuable lessons you can learn, fall under the category of "What not to do". And here, recently, I learned several (read that to be MANY) painful, yet valuable lessons about animation.

Armed with copies of ManCandy FAQs, Intro to Character Animation and Animating with Blender, I decided I was ready to attempt my first original character animation. (Yeah, that was probably my first mistake, over-confidence.)

Okay, so it seems there are three stages to creating an animation:

- Pre-production (Planning)
- Production (Do-it)
- Post production (Polish it up and finish)

Seems simple enough.

I normally do the greater majority of any project planning in my head. Little did I know that this would be just the beginning of my "What not to do" lessons.

After a few weeks of serious mental planning, I decided I was ready to start. Because I rarely write down more than a few notes about a planned project, first thing I actually did was design and model my characters. Now that in and of itself wasn't that much of a show-stopper. At that point, I could still have gone back and written a script, done storyboards and created the animatic.

Instead, I started creating my set. Now, I only intended to create a rough set, then I was going to get on with the whole “script/storyboard” thing. But well one thing led to another and before I knew it the set was built, textured and lit appropriately.

About this time, it is starting to dawn on me that I might really need the script and story boards. But I had planned a really simple story line, so I started rigging my characters instead. After several mis-tries, I got my characters adequately rigged. They weren't amazing rigs, but they did what I wanted. Which really is kind of the whole point.

Now, back to that dang script and story board. Well, I did finally write a short script and started making some storyboards. I ended up drawing a grand total of seven storyboards.

That should be enough, right? :P

Now, on to animating. I only need my characters to actually walk a very short distance. So first I tried manually keying my characters walking about the scene. But even when I drew out a path (with the grease pencil) they looked like drunken sailors careening about the scene. After numerous failed attempts to get my characters to behave, I did some research and moved on to creating a repeating walk cycle and then had my character follow a curve through the scene.

WOW!, that looked great. Okay a little on the stiff side, but better than the drunken sailor look. I might just be a little good at this :P. Besides I can always polish it up later.

So next, I merrily moved on to creating some other simple actions. Armed with my additional actions, I pulled up some tutorials on the NLA editor and prepared to combine and layer my actions for my animation. It was time to put this thing together.

Yeah that didn't work out so well. Every time I added a new action to one of my characters, it “teleported” to a completely un-related spot in the scene. Something was seriously wrong and or my project was haunted.

After many lengthy sessions filled with muttering of choice naughty words, I finally did get things to co-operate (kind of). But in the process, I managed to miss my deadline. Then of course, life interfered and my project got put temporarily on hold.

But even though I wasn't actively working on it, I was still obsessing about it. It took a few weeks, but the epiphany finally hit (Yeah, I'm a little slow, it's not like Roland didn't warn about these very types of problems in just about every chapter of his book).

My problems all stemmed from lack of proper planning (okay and maybe a few "lack of knowledge" problems, but I actually figured those out on my own).

If I had taken the time to go through pre-production properly, a lot of the problems I ran into could have been avoided. Creating a proper storyboard and animatic would have helped me block out my actions better (no drunken sailor walks). And, probably even more important, breaking it up into several shots would probably have eliminated most of the random character “teleportation” problems I ran into while combining my actions.

Elimination of those problems would have saved me time. I might even have had enough time to finish by my deadline.

Oh well, at least I have learned the importance of good pre-production.

Well that's all for now, I still have an animation to finish, so I should get back to it :P 🟥

Blender Conference 2009, 23-25 October Amsterdam

The 8th annual Blender Conference in Amsterdam will tentatively take place at a new location: the former 19th century court building "De Balie". Right in the center of Amsterdam on the Leidseplein, this grand cafe / theatre / cinema, sits nestled in the heart of Amsterdam's music, theatre and clubbing nightlife.

debalie

centrum voor cultuur en politiek centre for culture and politics

Due to the increasing number of attendees, a bigger place was needed. The De Balie offers a cinema / theatre with 160 seats, the smaller "Salon" with 50 seats, and sufficient informal space to meet, relax and lunch in our own grand cafe.

De Balie offers great multi-camera streaming services, and stores most talks on their servers for later viewing.

Call for participation

Deadline for paper / presentation / workshop submissions: August 1.

As usual we welcome Blender developers, artists, educators, professionals, scientists, to present work they've completed with Blender, to show great new ideas for Blender or propose to do round-table discussions, workshops, demos or courses.

Needless to say, special attention will go to the results of the Blender 2.5x project.

Submission details will be posted in May.

Suzanne Award Festival

"De Balie" has a real cinema with excellent projection facilities. We seek submissions this year to assemble a great 90 minute Short Film festival, which will run on three evenings, also for non-conference attendants.

Deadline for Festival submissions: September 15.

Submission details will be posted later.

Parallel sessions / conferences

In past years we've had parallel sessions organized by Blender community members in many cities world wide. If you like to organize a day (or two), contact ton at blender.org!

2.49: One More Version Before 2.5

Ton Roosendaal writes:

On popular request (users banging on our doors!), but mostly because current sources have seen much improvements, we challenge fate by releasing the very last available number before 2.50.

On Blenderartists a contest for the splash already started:

http://blenderartists.org/forum/showthread.php?t=152631

Features of 2.49 include:

- Video Textures in Game Engine
- Logic API cleanup for Game Engine
- Texture node editing
- Etch-a-ton, armature sketching preview
- Dome rendering for Game Engine
- Projection texture painting
- Jpeg2000 support in sequencer
- New GE actuators

Most of the branches remain 'branch' for the time being. So no volumetrics, no BMesh, no new Nurbs, no Freestyle, although all these progress well. Dome rendering for the GE we look at to include though.

-Ton-

We are currently testing the first Release Candidate for 2.49.

Download.blender.org/release/Blender2.49RC/ ■

By Byron R. Kindig

Introduction

The other day I was looking through some of the old family photos. Many of the people I have not met, or really even know who they are, except for a few notes that were hand written on the back of the pictures. I got to thinking about how I might animate them using Blender, and that is how this tutorial came about.

In this tutorial you will learn how to :

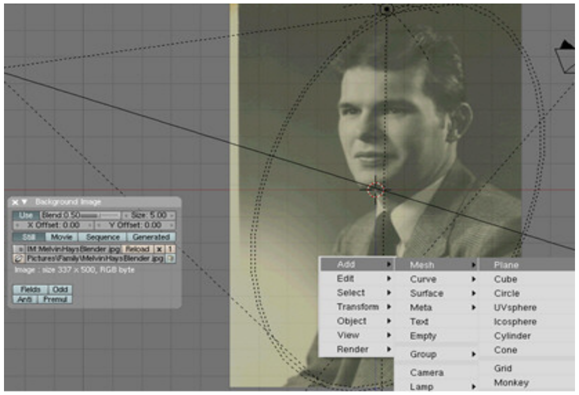
- Use a background image for modeling
- UV texture map a simple object
- Set up Shape Keys
- Animate Shape Keys

This is the picture I decided to use, but of course you can use one of your own family photos. If it is not in a digital format you will have to scan it and save it to a resolution of about 500 to 800 pixels in the longest direction.

After deciding on an image, open up Blender and open up the image in the background. To open a background image, in the 3D view port footer (or header) click the “View” ›› “Background image” ›› “Use Background image” ››“Load.” Find and select the image you want to use in the resulting menu, then click “SELECT IMAGE” at the top of the menu.

It may not appear immediately. If not, be sure you are in a direct view such as front, top or side. Remember the final object is a flat 2 dimensional plane like the photo you are animating, so you only need one 3D view. I used the front view.

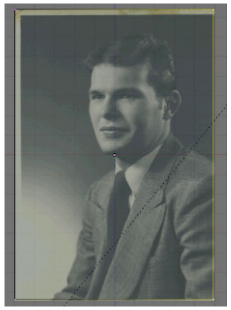
Add a Plane to your scene with [Spacebar] ›› Add ›› Mesh ›› Plane. Toggle to Wire Frame view with the [Z] key so that you can see your Background Image.

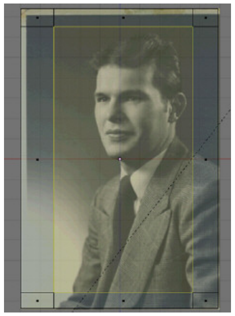

Background Image
Use | Blend 0.50 | Size: 5.00
X Offset: 0.00 | Y Offset: 0.00
Still | Movie | Sequence | Generated
IM:MelvinHaysBlender.jpg | Reload
Pictures\Family\MelvinHaysBlender.jpg
Image : size 337 x 500, RGB byte
Fields | Odd
Anti | Premul

In Edit mode, move the corner vertices to match the corners of your background photograph. You can deselect all the vertices with the A key, then use the B key to border select the top 2 and the [G] ›› [Z] keys to constrain the movement to the Z direction. Then "Swap Select," [Ctrl + I] then [G] then [Z] again to move the others to the bottom of the photo background image.

by Byron R. Kindig

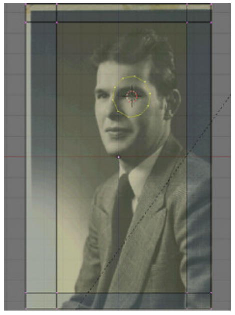
Border select the left vertices, press the [G] and the [X] and move them to the left side of the photo background image, and then swap select, [Ctrl + I] and move the right hand side of the background image.



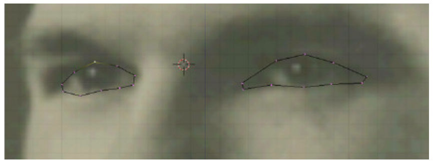


The next step is where we will create the vertices that will outline the facial features that we will animate. This can be done in lots of ways, but basically you will have to create loops of edges around the eyes and mouth. You can then extrude these edge loops to cover larger areas of the face and then fill in the faces to make sure the entire plane is filled with faces.

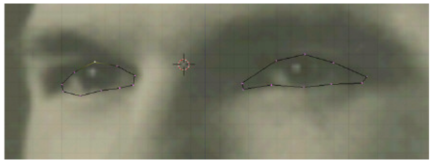
Start by adding a set of Loop cuts around the edges of the plane to cut out some of the non-animated area around the head and body. [Ctrl + R] ››, LMB Slide and LMB. Then delete the face inside this border area by selecting it in face select mode and pressing [X]›› “Face Only”.



Still in edit mode, but changing back to vertex select mode, add a circle with 10 vertices for the loop around the eyes. Scale it down close to the size of the eye and move it to the area near the eye. Zoom in so you can see what you are doing with the middle mouse wheel. Then move the individual vertices of the circle to the corners of the eye, and to follow the outline of the eye.

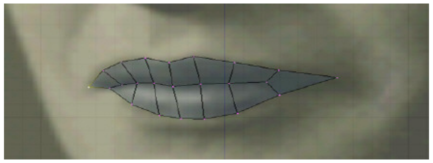


Once you are satisfied, duplicate the circle by selecting one vertex of the circle, [L] ›› [Shift + D] and move it to the area around the other eye on the photograph, then adjust the vertices to fit the second eye. Then select the upper vertices of the upper eye lid and extrude [E] ›› Only Edges and scale the vertices on the top of each eye upward to just under the eye brow and again to just over the eye brow.

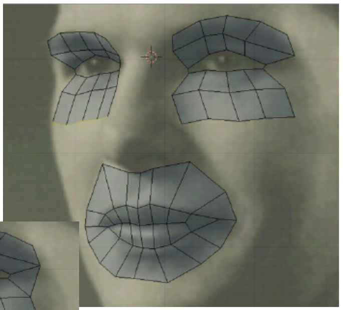


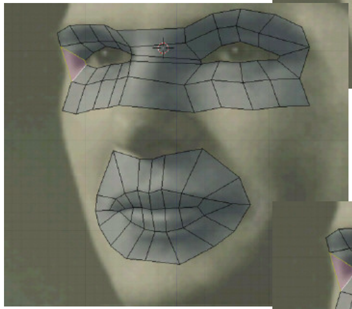
Un-select all the vertices with the A key once or twice and then [Ctrl+LMB] click on the center of the upper lip, to add a new vertex to the mesh. Continue to [Ctrl + LMB] click around the mouth to make a full circle around the lips. Then add the vertices in the middle where the lips come together and fill with faces by selecting 4 vertices or 2 edges and pressing the F key.

by Byron R. Kindig



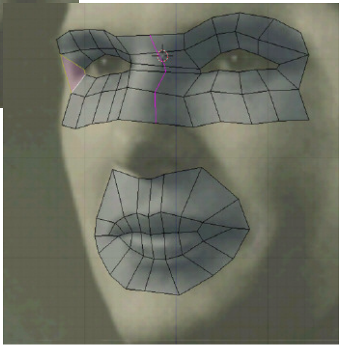
Continue to extrude and shape these edges and fill in the faces.



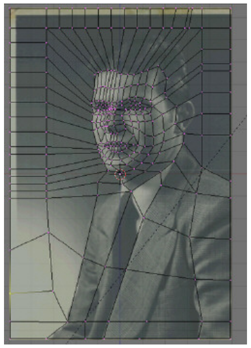


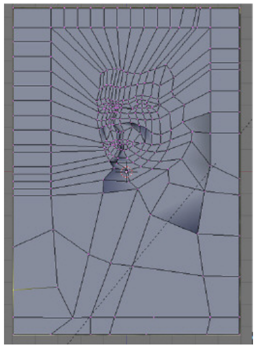
Loop cut the longer edges and arrange the vertices, until the area is filled.

It should look similar to this.



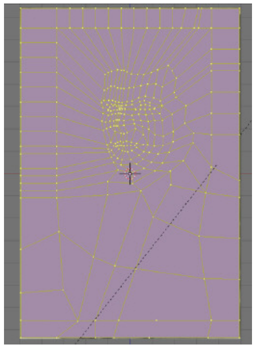
Fill in the faces in the holes left in the areas where the eyes are. Select each pair of upper and lower eye lid edges and press the [F].





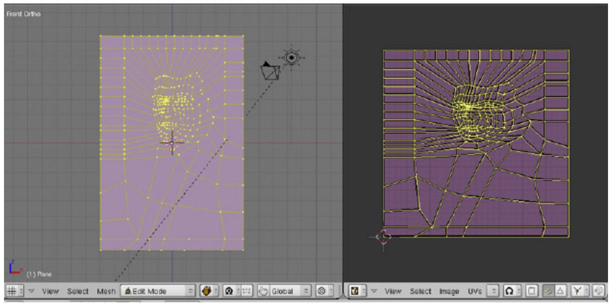
Now return to solid view by toggling the Z key. If you notice some of the faces are shaded strangely, like this.

Then you will need to recalculate the normals outside by selecting all the vertices and pressing the Ctrl + N key. It should then look something like this.

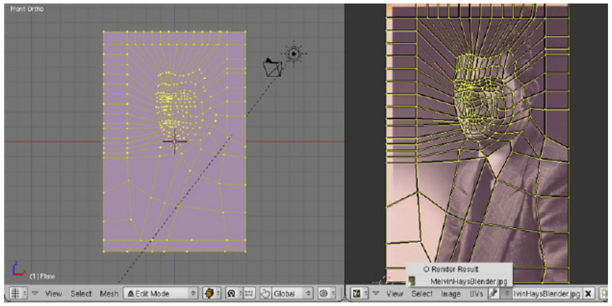


by Byron R. Kindig

Next split your screen and add a UV image editor window on the side. With all the vertices still selected and the mouse in the 3D view port window, press the U key and select the “Project from view (Bounds)” option. It will look stretched out from side to side, but don’t worry about that now.



As soon as you open the image from the menu it will snap to the right dimensions.



In the shader buttons add a new material and name it UV. Add a Texture and make the type “image.” Choose the same background image from the menu. Back in the shader buttons Map Input panel, change “Orco” to “UV.” In the Material panel, select “TexFace” and in the Shader panel, move the Specular slider down to 0.00.

Change the View port Shading to Textured. You should see the image mapped to your mesh at this point. If not, check the back side by rotating your view or pressing [Ctrl + NumPad 1] to see the back. If it is on the back side, return to the front NumPad 1 and with all the vertices selected, press the "Flip Normals" button in the Mesh Panel of the Edit buttons. This will solve the problem if it exists.

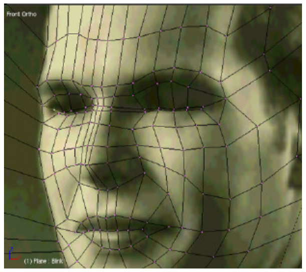
Now you can add shape keys to the plane mesh and it will deform your photograph that is mapped to it.

Warning: You will not be able to add or delete any vertices from your plane mesh after the next step.

In object mode in the edit buttons in the Shapes panel, press the “Add Shape Key” button. Blender will add the first shape key and name it “Basis.” This is the only shape key that does not have a control slider and is the reference for all the other shape keys. Add another shape key and it will be named “Key 1.” We will use this key to close the eyes so rename it “Blink.”

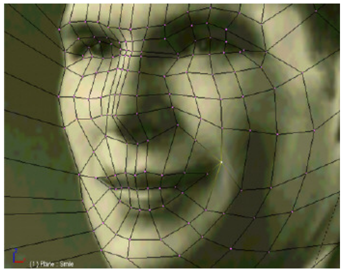
Zoom into the eyes in the 3D view port window. Tab into edit mode again, and select each upper eyelid vertex and with the G key move each one down to its corresponding lower eyelid vertex. Now return to Object mode with the Tab key and try out the slider for your newly created “Blink” shape key. Note that you can move the eyelids along a continuum from all the way open at 0 to all the way closed at 1, or anywhere in between.

by Byron R. Kindig

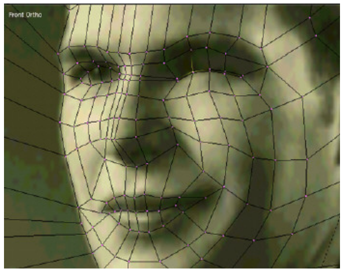


Before adding your next shape key be sure to return to object mode if you are not already there, and to the Basis key. If you were to add another key with the "Blink" key selected it would inherit the eyes being closed and since you are going to add a smile shape key next you probably don't want to have your photo close its eyes each time it smiles.

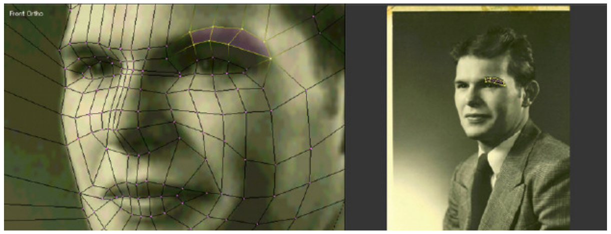
So in the Object mode and with the Basis key selected add another shape key and name it "Smile." Return to edit mode and move the vertices around the mouth into a smile. Remember to also move the vertices at the corners of the eyes into a little bit of a squint and adjust the vertices of the cheeks as well. Go back to object mode and try it out. Remember if you are not satisfied with a shape key you can return to edit mode and make whatever changes you want.



Return to Object mode and to the "Basis" shape key and another shape key and name it "Wink." A wink is different than a blink in several ways. Most obviously, it only is on one side of the face. Less obviously, the lower eyelid goes up a little and the upper eyelid comes down a little instead of only the upper eyelid coming down to meet the stationary lower eyelid. This can be accomplished by selecting the edges between the upper and lower eyelid vertices and scaling them down. Also the corner of the mouth on the same side raises and the cheek on the same side is deformed slightly. Even the area where the nose meets the cheek goes up a little.

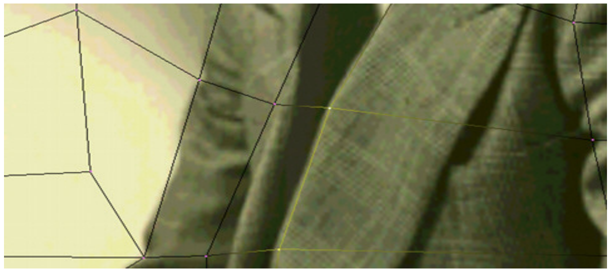


Let's add another shape key for the Left Brow Raise. Remember to return to Object Mode and the "Basis" key and then "Add Shape Key" and name it "LBrowRaise." In edit mode simply adjust the position of the vertices around the images left eye brow. I am using the left because that is the side that appears closer in the image I am using. Your photo may be facing to your right and you may want to use the other eye and eye brow. This can be very subtle and still very effective.

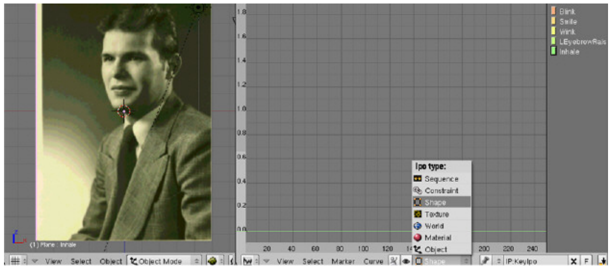


For the last shape key we will move down to the left lapel of the jacket and make another very subtle adjustment to show the movement for inhaling.

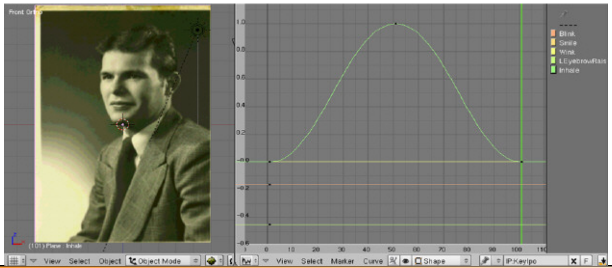
Create a new shape key in Object mode from the "Basis" key and name it "Inhale."



Now for the next section you will add these shapes to an IPO to create your animation. Make the window on the right into an IPO window. Make the IPO window type "Shape."

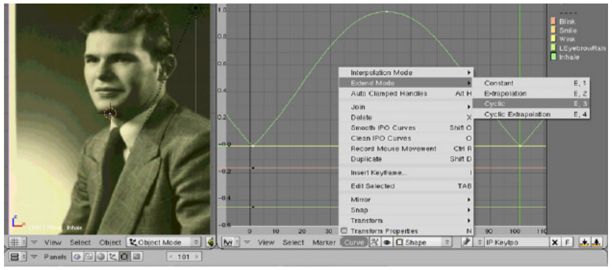


Set a key frame at frame 1 for the "Inhale" shape at 0 on its slider. Move the frame number forward to frame number 51 and add another key frame with the slider

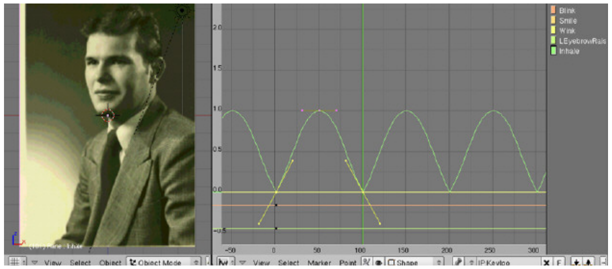


set to 1, and another with the frame number set to 101 and the shape slider set back to 0.

Right click on the Inhale curve to select it. The key frames will turn white. In the IPO footer click on Curve then Extend Mode and then Cyclic, to make the inhale curve repeat. We will render our animation for 300 frames so the breathing will loop continuously.



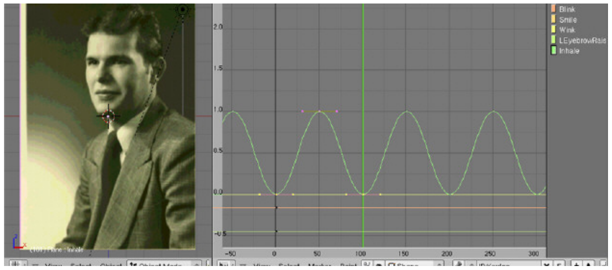
Unfortunately, you may notice the bottom of the curve where it repeats has become rather sharp and pointed instead of smooth as we wanted for a smooth breathing cycle. Tab into edit mode with the mouse pointer in the IPO window and select the 2 bottom control points of the curve.



by Byron R. Kindig

by Byron R. Kindig

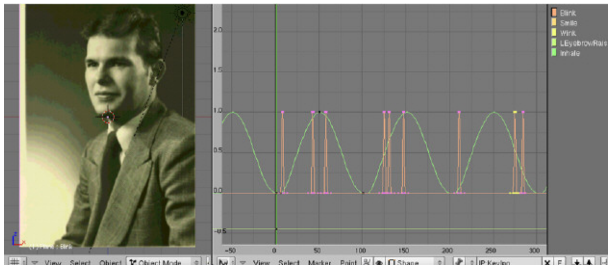
You can smooth the curve with [Shift + O]. Now it is smoother and looks like we want



Go to the Scene buttons (F10) and set the end of the render to 300. With the mouse pointer in the 3D view port window, press Ctrl + A to play your animation and watch your photo breathe! It will loop until you press the Esc key. When you are done watching it press the Esc key and we will add some more shape IPOs. Next we will add the blinks into the animation.

Return to the Edit buttons (F9) and in the Shapes panel, choose the “Blink” shape. Set a key frame at frame 6 and the “Blink” slider at 0. Move forward to frame 8 and set another key frame with the slider at 1 and another on at frame 10 again with the slider at 0. With the Blink IPO curve still selected, tab into edit mode. Remember to keep the mouse pointer in the IPO window.

Select the 3 control points of the curve with the border select tool, [B]. Press the [Shift + D] to duplicate the control points and then hold down the Ctrl key to constrain the duplicated key frame and control points to the even frame numbers and slide them over to a new location. Repeat this duplication for several more blinks during the 300 frames of the animation. Allow the spacing to vary as blinking is much more random than the breath cycle that we created earlier.



Tab back out of edit mode (with the mouse pointer still in the IPO window.) Create another IPO curve for the Wink shape, by adding a key frame with the slider at 0 another at 1 and a last one at 0 again. In edit mode you can move the key frames forward or back to adjust the timing. Remember to use the [G] and [X] to move along the X axis of the timeline and to use [Ctrl] to constrain to an exact frame number.

Do the same with the smile and the eyebrow raise shapes. Remember if vertices are used in more than one shape and the shapes are applied at the same time, the movement of those vertices will be added. In other words if a “Blink” is on the same place on the timeline as the “Wink” then the Winking eye's vertices may overshoot their mark.

Also if you forget to hold the Ctrl key while moving the control points of the IPO curve and they fall in between frames this can cause problems. You can select the control points, either individually or all of them with the A key, and use the snap menu to bring them back to whole frame numbers. [Shift + S]›› “ To Frame”.

If you notice that the breathing is a little too exaggerated, in edit mode, you can select the control point set to a slider value of 1, in the IPO curve and lower its effect by moving it down with the G key followed by the Y key to constrain it to the current frame.

by Byron R. Kindig

Because it is just one breath cycle, set to cyclic so all the other breathes will be adjusted at well.

Next we will set up our lighting and camera. You do not need any shadows since you are rendering a flat plane with a photo that has its own shadows recorded. You only need one light source, a Sun lamp so that the Plane is evenly lit. You also don't want to have perspective as it would only serve to distort your image. Select the plane and snap the 3D cursor to it [Shift + S] ››" Cursor to Selection." Select your camera and snap it to the 3D cursor, [Shift + S], ›› "Selection to Cursor."

With the camera still selected, in side view, move the camera out in front of the plane with [G] ›› [Y] to constrain the motion along the global Y axis. Press [Alt + R] to clear the rotation of the camera. Now rotate the camera 90 degrees around the X axis by pressing [R] ›› [X] and typing in the number 90, then press [Enter] or LMB click.

Switch the 3D view to camera view with [Numpad 0]. In the Scene buttons window, (F10) in the format panel, set the Size X and Size Y fields to the size and proportions you want to render your animation. I have set them to X=400 and Y=600, because those are the dimensions of my photo.

Now switch back to the edit buttons. In the Lens panel click on "Orthographic,"and adjust the Scale field in the Lens panel to a size that includes your plane in the camera view. I have set mine to 15 but depending on your photo you may need to change this.

Next render your animation. Back in the Scene buttons window. (F10), check to be sure that in the Render panel, the Shadow, SS, Env M, and Ray buttons are turned off. You will not be using them and they will only slow down your render.

Click the "ANIM" button in the Scene buttons window. (F10) and wait for the resulting animation.

I hope this has been a helpful and most of all fun ■

Sincerely,

Byron R. Kindig

By Valentin Spirik

Introduction

For everything you need to get started with the Blender VSE see also the "Video Editing Preset" (and for making titles) the "2D Title Presets" .blend files bundled with this edition of BlenderArt Magazine.

I load all of my footage into the Blender Video Sequence Editor (VSE) and start selecting "the good stuff": with the mouse over the Image Preview window. I use [Space] for start/stop and [Right Arrow]/[Left Arrow] for going back and forth. In the timeline (= Sequence): [K] for Cut and [Shift S] for snap editing (shortens/extends

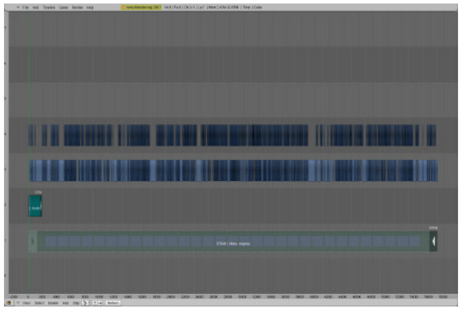


Figure 1: Selecting "the good stuff"

clip to the playhead when start/end is selected). The "good stuff" gets moved upwards a row (= Channel) with [G] for Grab and [Y] for Y-Axis. I keep the uncut original inside a (muted) Metastrip should I need it later since I am working without a time-code. (You can use the Stamp render option for overlaying time information.) Then I make a back-up of the .blend.

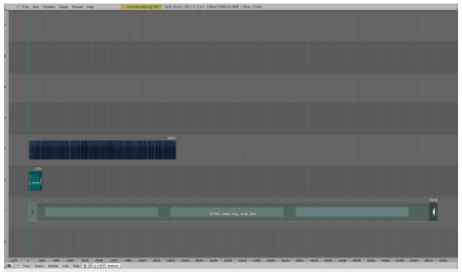


Figure 2: Selected vs. original footage

I move the selected clips together using [Ctrl] for snapping clip to clip. The white numbers on the right show the last frame of "the good stuff" (top), the music track (middle) and the unused clips etc., that are now also inside the muted Metastrip (bottom).

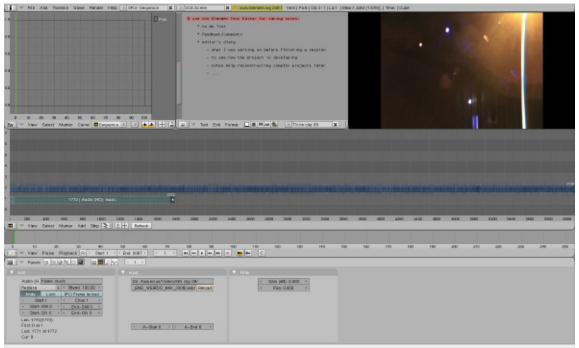


Figure 3: Selecting "the really good stuff"

by Valentin Spirik

Same procedure as above (1: Selecting "the good stuff") with the mouse over the Image Preview window (top right) and [Space] for start/stop and [Right Arrow]/[Left Arrow] for going back and forth... Since I easily get real-time playback using DV footage (no effects applied) with Ubuntu 8.04 and Blender 2.48a on my PC (current Core 2 Duo processor, 2 GB of RAM and a reasonably fast 250 GB hard-drive) this work-flow allows me to find my in and out points while looking at moving images the way the audience will see it... Frame 6 067 is now the last one after "the really good stuff" is moved together (that's down from 20,520 from the previous step). I use the Blender Text Editor for taking notes. Again, I make a back-up of the .blend.

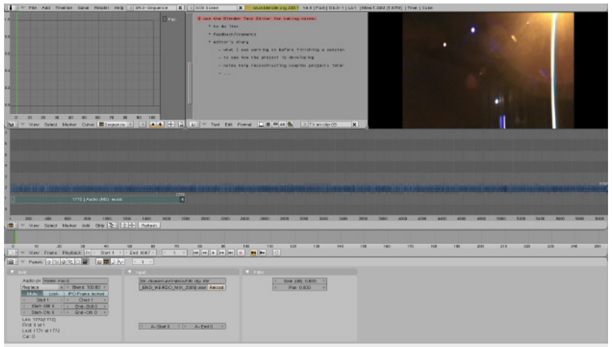


Figure 4: Sorting the clips

I add a Text Editor window left of my Sequence and name/tag my tracks. I adjust the Sequence window with (Number Pad) [Home] and then use Middle Mouse Button and [Ctrl] to fit the Channels to my tagged tracks. Then I move "the really good stuff" clips in the appropriate track (for moving clips up or down without moving them in time I once again use [G] and [Y], for selecting multiple clips I use [B] for Border Select...).

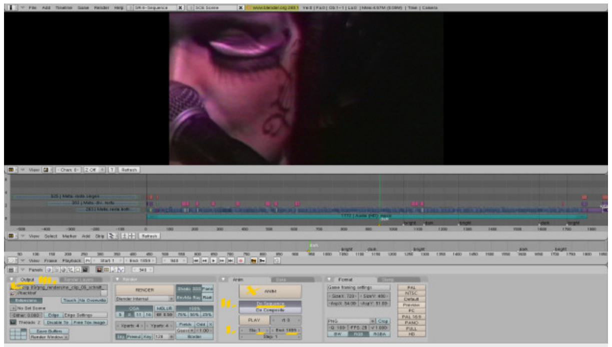


Figure 5: Editing the video

I use Markers for the points where the singing starts. Then I move the best of the previously sorted clips to roughly where I need them and work on the details again using Shift S for snap editing and also [G] for Grab to shorten/extend a clip's start/end... The basic structure for the edited video: details/close-ups at the beginning, the artists performing in the middle and a longer zoom out close to the end. I move the unused clips into Metastrips on the left.

As before, I use (Number Pad) [Home] for seeing everything that's in the timeline. But since there are the unused clips on the left now, I then select the music track and (Number Pad) [Dot/Del] (normally used for "zooming in" on a single clip) for focusing the Sequence window to the area that I actually want to see when editing...

I export the edited clip as a PNG Sequence (= series of .png images). (F10 (pressed multiple times) for switching between the Sequencer buttons when editing and the Render buttons when exporting.)

The annotations in the screenshot above show what's important for exporting/rendering.

by Valentin Spirik

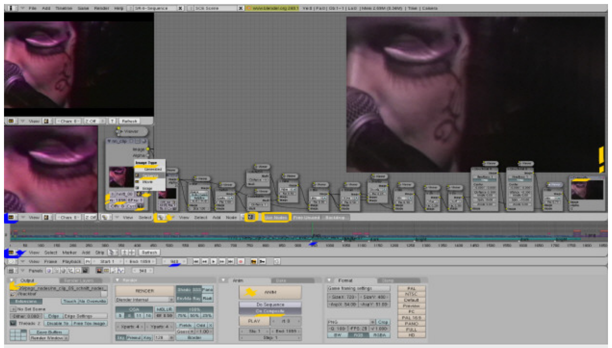


Figure 6: Through the Compositing Nodes

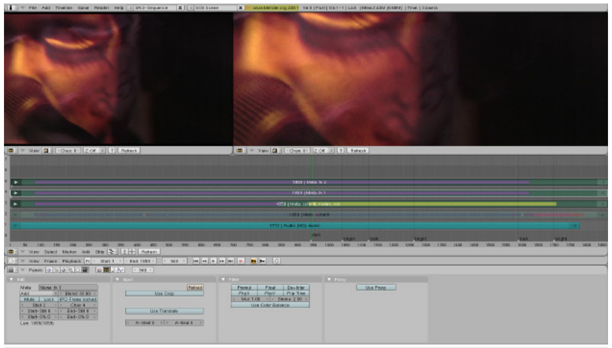


Figure 7: The final look

I load the previously exported PNG Sequence into the Compositing Nodes and let it run through a series of filters: DV video artifacts get smoothed out and my own graininess/structure and artifacts get added. The particular filters used here may be experimental and only make sense for this clip, but a couple of combinations can be useful for other projects (values would need to be adjusted): Gauss-Darken for a toon look, Screen-Overlay-Mix for optimising an image, Sharpen-Soften for focus related compositing tasks (high values for Sharpen may introduce artifacts).

The yellow and blue annotations in the screenshot above show what's important for processing a series of PNGs with the Compositing Nodes: in this set-up, navigation is possible by moving the Sequence playhead (the green line at frame 940) for previewing different parts (individual frames) of the Nodes processed video. I export the Nodes filtered PNGs once more as a Sequence of PNGs.

I load the Nodes processed PNGs into my VSE Sequence ([Space] ›› Add Image Sequence), add and fine-tune a Glow (with clip selected [Space] ›› Add Glow), make a Metastrip of both (with clips selected and [M] and use Color Balance (with clip (Metastrip) selected: Filter tab ›› Use Color Balance) for creating the basis of the yellowish (but at this point rather dark) look. I then add a one step Strobe (Filter tab ›› Strobe: 2.00). With [Shift D] I duplicate the Metastrip, move it up one Channel and also move it one frame out of sync (see screenshot). The Glow for this FX 1 track gets removed ([Tab] to open/close a Metastrip) and in the Edit tab I select Blend Mode Add and Blend: 33.00 and Use Color Balance gets also unselected for this FX 1 track. I also duplicate the FX 1 track, move it up one Channel and also move this FX 2 track out of sync for yet another frame. There is one last step not shown in the screenshot: a final, subtle Glow is applied to all strips.

I once more export the clips as a Sequence of PNGs, import them one last time and change the export settings in the Render Buttons ›› Format tab to FFMpeg, make my selections for the video and audio format/codec and make sure that Multiplex audio is selected in the Audio tab before exporting the finished video clip with sound.

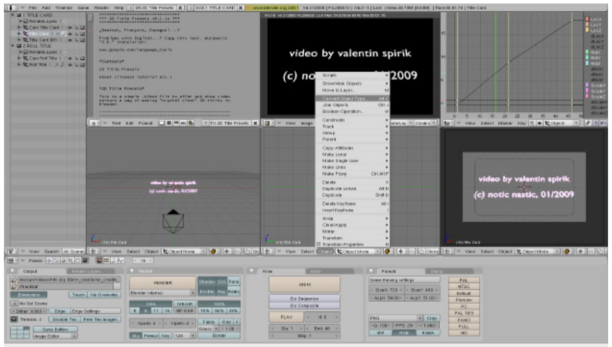


Figure 6: Through the Compositing Nodes

This is actually done somewhere before finishing the editing (5: Editing the video) and using my own 2D Titles Preset .blend/tutorial. I adjust the text to the right size first. In order to have the Center where I need it for my title animation: Object ›› Convert Object Type... ›› Mesh. Then: Object ›› Transform ›› Center New.

For the finished video see indiworks.blip.tv, for more about Notic Nastic see www.noticnastic.com 🟥

by Valentin Spirik

Product Visualization with GLSL and
Pre-Rendered Environmental Materials

By Claas Kuhnen

Introduction

Fast rendering times are always a needed and desired feature in a time wise very competitive environment like industrial design. It is very common today that the product designer only makes very rough renderings, which are being presented to marketing, while the 3D rendering department or an outsourced service bureau is producing more realistic, but also time consuming product representations.

For very fast and preliminary product representations, the designer only needs to communicate material selection and shape to marketing. Those product renderings give answers as to which part of the product is high glossy, which part is chrome, what is rubber, where is maybe an interface element or decal. Thus an integral part of product rendering is the decision making about material selection.

In the last 2 years, Hypershot has become a product of choice for such a task. It provides the artist with very fast previews and instant feedback. The progressively refined renderings are done with global illumination and thus provide very fast, accurate and good looking product render results.

Blender, of course, does not offer an internal full GI or a fast progressive render feedback like Hypershot. However with Blender, we can create graphically stunning looking product representations, which are ideal for feature communication and blazing fast to render for animations. In particular the last one - animation - is where Hypershot lacks completely and Blender can provide an excellent number of tools.

What I am going to show is actually inspired by a free tool for Rhino 4.0 called Auxpecker, which gave me the idea to experiment with this approach in Blender with GLSL. http://auxpecker.blogspot.com/

2 Light-Setup:

While Blender does not have an internal full GI system, it offers real-time shading with GLSL and very fast rendering Approximate Ambient Occlusion. GLSL, which was mainly driven through the last open-game project Yo-Franky, provides us with the needed technology to actually preview diffuse and specular material properties interacting with many built-in light types and can also be used with UV mapped and flat projected image textures.

A great feature of GLSL is the ability to show diffuse and specular light shader properties in real-time, interacting with the present set of lights. There are two lights which in particular are of interest:

- Hemi Lights - Producing a fake, but still good looking and fast rendering GI effect.
- Spot Lights - Casting buffered shadows in real-time

Other light types are:

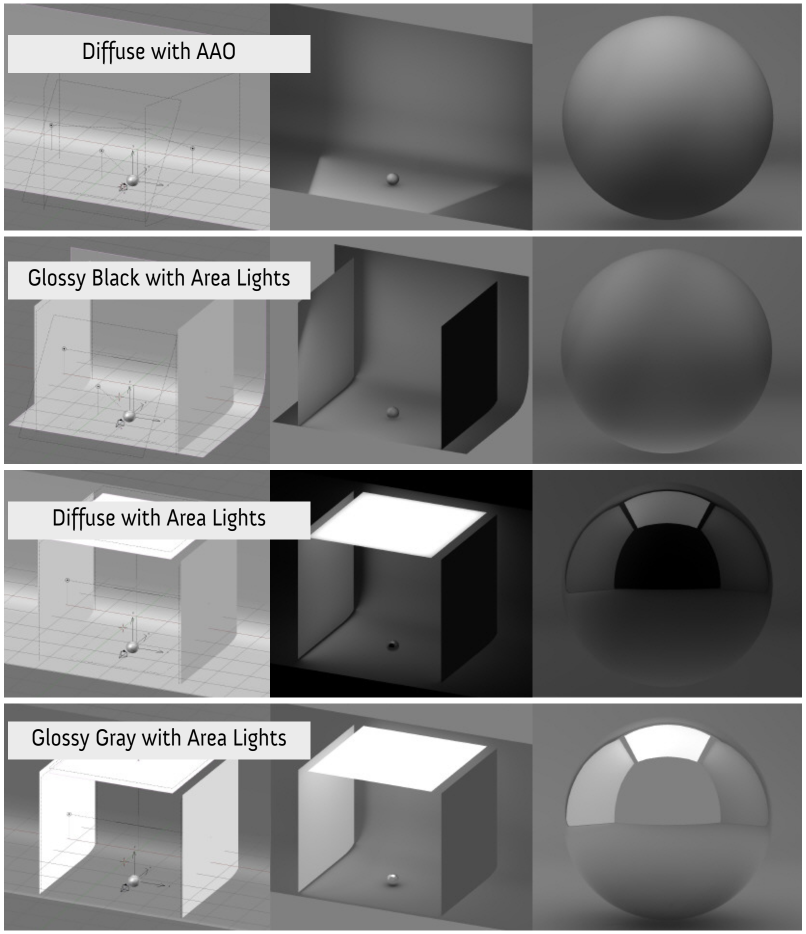
- Lamp Light - Producing a spherical light element
- Sun - Producing parallel light rays

Area lights are not supported for real time previewing, but work during rendering.

by Claas Kuhnen

3 Material-Setup:

To tackle the issue of evaluating a material without time consuming raytraced mirror reflections, we can actually generate pre-rendered material previews. They are in the shape of spheres and are mapped as a fake environment over the object by using the Normal Vector map input.



The clue is to prevent the use of any easy to recognize environmental mirror reflection, instead creating a studio setup which only provides information about where light sources are and how they would illuminate the scene and the ball.

You design a simple backdrop, place a ball, and position the camera very close to the ball, trying to find a scale for the mesh where the mirror reflection of the ground plane would be covering the lower half of the ball.

With this setup you can add lights. Three area lights are sufficient. They are placed left, top, and right. They can all have different energy values, thus rendering the ball illuminated in a different and not like with AAO even way. To generate unique results, you can play with size, position, and strength of those lights and create materials where a strong light is only coming from top, the main light coming only from the side, or light is coming from all directions evenly.

To make the area lights visible on a chrome ball, you also need to add light meshes. By applying correct values for the Emit value, you can match the brightness of those meshes to the light energy of the corresponding lights.

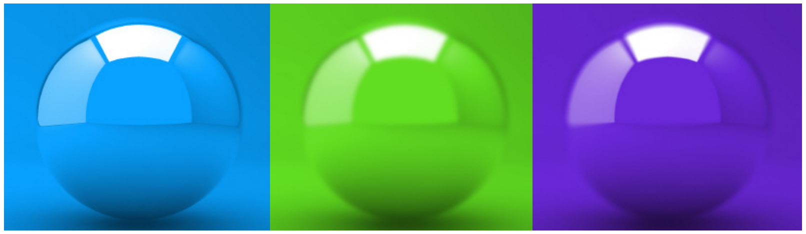
By using AAO, you can darken the area between sphere and ground plane. An up facing area light can also brighten up the lower part of the sphere simulating indirect illumination and bouncing lights.

If a glossy material is desired, the sphere should be using a mirror reflection and the scene should have the visible light meshes be present. Those are being used to generate a specular highlight instead of using the specular light shader, which I turned off in this case. Again with the different amount of emit values, the reflections of those boards will be differently bright and providing a convincing studio setup.

For a black material, the world should also be set to black, darkening the mirror reflection of the ball. For a red ball, the world should be set to neutral gray. This way the rendered result can easily be coloured in Photoshop to generate quickly colored variations of the same gray base material.

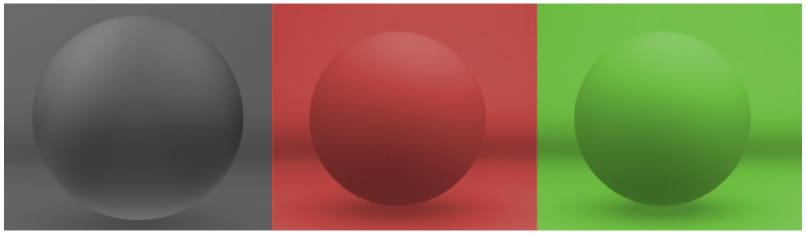
By using the raytraced function for mirror reflection, you can also make use of Fresnel to customize the reflection value on perpendicular and tangent surfaces relative to the camera.

Of course to create blurred reflections or brushed metal, it is also possible to use Blender's glossy function for mirror reflections. However to save time, this task can easily be done inside Photoshop by applying a blur filter to the image.



Colored Colored and Blurred Colored and Lens Blurred

If a diffused and matte surface is needed, then the specular reflection will be used instead of raytraced mirror reflections. The energy is set low and the hardness value is set to very low to spread the specular highlight nicely over the object. It is also possible to make use of Blender's subsurface scattering shader to create a more rubber looking material.



SSS - Raw Rendering Colored Photoshop Material

Also pay attention to the design of the scene and how the backdrop itself is part of the reflection. Scale of the sphere and reflection of the environment can have a significant impact on how light, shadow, and reflections are rendered and thus forming the material character.

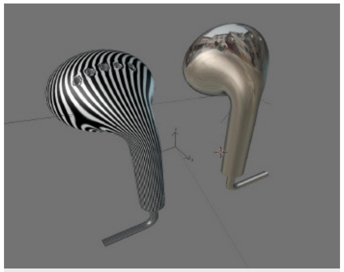
by Claas Kuhnen

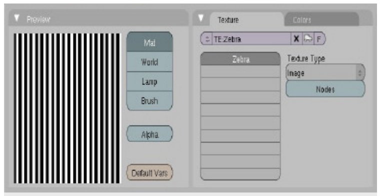
by Claas Kuhnen

Surface Quality:

To analyze the geometry and surface quality, in particular for sharp edges, uneven elements, or how a highlight is broken over an edge, it is also possible to map an environmental image or a zebra image over your object.

The Environmental image basically makes the object look like a strong chrome ball, while the Zebra image produces vertical lines which then when the object is being rotated will move over the surface. Any uneven parts in that Zebra pattern will identify a problem with the geometry.

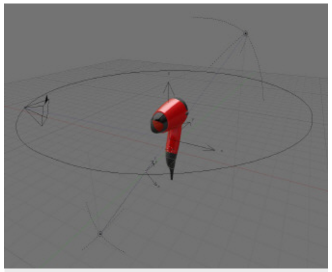

GLSL Preview showing distortions

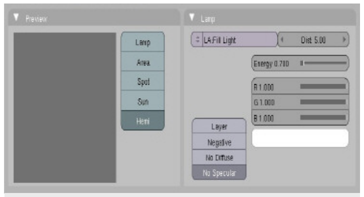


Zebra Image Texture

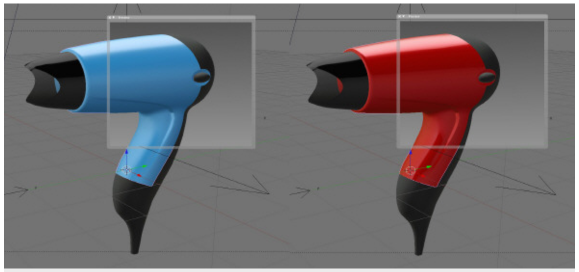
Application:

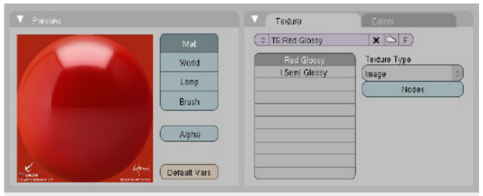
Personally, I prefer using the Hemi light approach. I use two opposite facing hemi lights. The top one is stronger and functions as the main light, while the second one can be used as a fill light and has a much lower light energy. Both lights have their specular function disabled and only add a little of additional diffuse value to the scene. The material illumination and highlights are already done with the pre-rendered material. Again, realistic light reflections are not equal to the specular light shaders we are used to. In nature the reflection of the light source is also the the shape of the visible highlight. This is why the mirror scenes have light meshes to generate those types of specular highlight reflections. With the lights having their specular value activated, the highlights would conflict with the rendered highlights of the material.


Turntable Setup with Hemi Lights

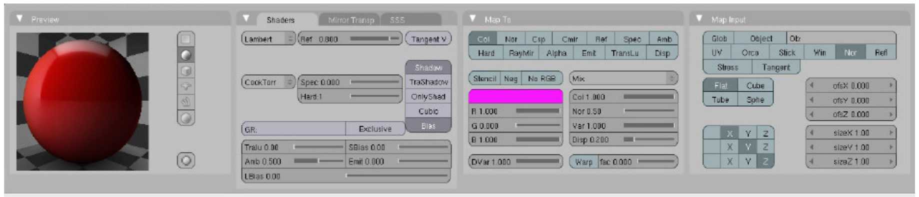


Hemi Light No Specular


Hair Dryer Blue Hair Dryer Red



Hair Dryer Red Texture

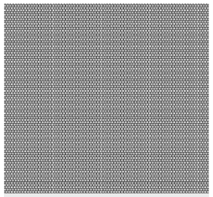


Lambert, Ref 0.800, Tangent V
CookTorr, Spec 0.000, Hard:1
Shadow, TraShadow, OnlyShad, Cubic, Bias
GR: Exclusive
Tralu 0.00, SBias 0.00, Amb 0.500, Emit 0.000, LBias 0.00

Map To: Col, Nor, Csp, Cmir, Ref, Spec, Amb, Hard, RayMir, Alpha, Emit, TransLu, Disp
Stencil, Neg, No RGB, Mix
R 1.000, G 0.000, B 1.000, DVar 1.000
Col 1.000, Nor 0.50, Var 1.000, Disp 0.200, Warp, fac 0.000

Map Input: Glob, Object, Ob:, UV, Orco, Stick, Win, Nor, Refl, Stress, Tangent
Flat, Cube, Tube, Sphe
ofsX 0.000, ofsY 0.000, ofsZ 0.000
sizeX 1.00, sizeY 1.00, sizeZ 1.00

Hair Dryer Red Material



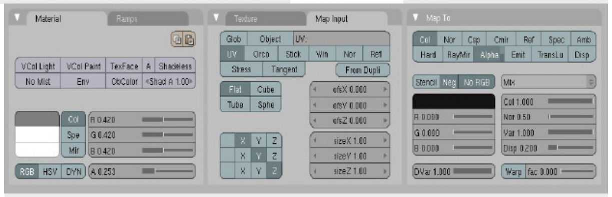
Screen Texture

Because of GLSL, I can instantly see how the diffuse value of the light is affecting my scene and I do not need to create a rough preview rendering. Colors you see with GLSL are colors you will also get rendered.

Another also very attractive and flexible approach, is to use the emit value for the applied materials, but set it rather low so the materials seems to just get a little bit of indirect illumination and then create with contrast rich lights, dramatic light setups.

GLSL can also nicely preview transparent materials. The screen of the ear bud is UV mapped with a screen material which is making use of the Alpha channel affecting the final alpha value of my material. The screen material together with another environmental image of a chrome surface produces the result of those metal caps.

Apple Earbuds Rendering - Time : 4 Seconds

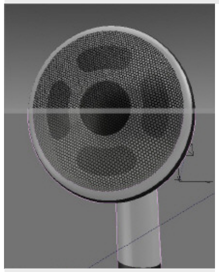


Material: VCol Light, VCol Paint, TexFace, A, Shadeless, No Mist, Env, ObColor, Shad A 1.00
Col, Spe, Mir: R 0.420, G 0.420, B 0.420
RGB, HSV, DYN, A 0.253

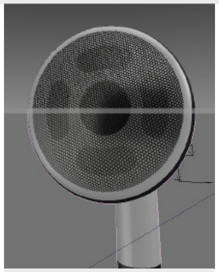
Map Input: Glob, Object, UV:, UV, Orco, Stick, Win, Nor, Refl, Stress, Tangent, From Dupli
Flat, Cube, Tube, Sphe
ofsX 0.000, ofsY 0.000, ofsZ 0.000
sizeX 1.00, sizeY 1.00, sizeZ 1.00

Map To: Col, Nor, Csp, Cmir, Ref, Spec, Amb, Hard, RayMir, Alpha, Emit, TransLu, Disp
Stencil, Neg, No RGB, Mix
R 0.000, G 0.000, B 0.000, DVar 1.000
Col 1.000, Nor 0.50, Var 1.000, Disp 0.200, Warp, fac 0.000

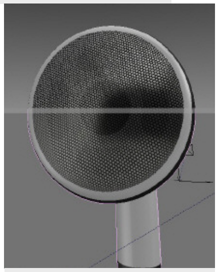
Screen Material



Alpha = 0.200



Alpha = 0.500



Alpha = 0.500

The presented rendering of the Apple ear bud only takes roughly 4 seconds. This is very important for any sort of animation. Because Blender comes, of course, with a rich set of animation tools, it is also possible to generate some stunning looking product visualizations in a very short amount of time.

The animations can include exploded views, best done with approximate ambient occlusion, light value

by Claas Kuhnen

animations like lights turning on and off, and of course simple camera path animations showing the product on a turn table.

Of course, is this not a substitute for full fledged Global Illumination rendering approaches. However where time is tight and results need to be communicative rather than realistic, the combination of tools Blender offers are a great time saver.

4 Conclusion:

In this presentation, I only covered the basic requirements for this work flow. With the progression of the GLSL shader development, more possibilities might become available. Currently GLSL can only show texture images by using UV and flat mapping excluding cube-clipping, which also limits this tool ability to preview in real-time graphic style elements because they cannot be projected onto the object by utilizing the empty as a projector.

However what we have currently is also probably not the final stage of GLSL.

So stay tuned ■

by Claas Kuhnen

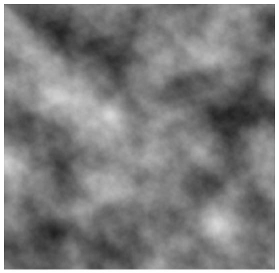
By Benjamin Schram

Introduction

In this Gimp tutorial I am going to show you how to create a nice eye/iris texture from scratch. In writing this tutorial, I used Gimp 2.4, but since it only uses basic filters and tools, any version of the Gimp should work.

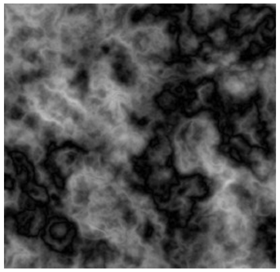
Step1

- Create a new image...
- Dimensions: 1024px by 1024px
- Rename the base layer to "Pinched".
- Add transparency...
- Layer -> Transparency -> Add Alpha Channel
- Apply "Solid Noise" filter...
- Filters -> Render -> Solid Noise
- Check: "Randomize"
- Detail: 15
- X & Y Size: 4.0



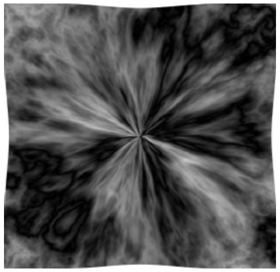
Step2

- Apply multiple "Difference Cloud" filters...
- Filters -> Render -> Difference Clouds
- Check: "Randomize"
- Detail: 15
- X & Y Size: 4.0
- Repeat filter [Control + f]
- Repeat 5-8 times



Step2

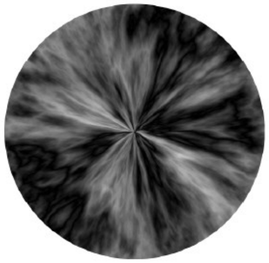
- Apply multiple "Pinch" filters...
- Filters -> Distorts -> Whirl and Pinch
- Whirl angle: 0.00
- Pinch Amount: 0.650
- Radius: 1.3
- Repeat filter [Control + f]



by Benjamin Schram

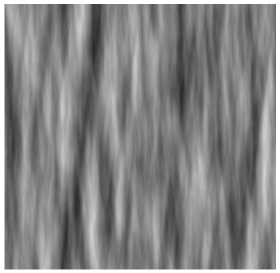
Step4

- Ellipse select on pinched layer...
- Use "Ellipse Select" tool [e]
- Make a canvas sized circle...
- Click at 0,0
- hold [Shift] to keep 1:1 proportions
- Drag to 1024
- Shrink the selection...
- Select -> Shrink
- Shrink: 20px
- Invert selection...
- Select -> Invert
- Clear background...
- Edit -> Clear



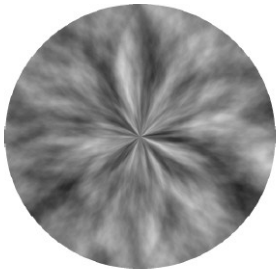
Step5

- Create a new layer...
- Name: Polar Coords
- Width/Heigth: 1024
- Fill Type: Transparency
- Select polar coords layer.
- Apply stretched solid noise...
- Filters -> Render -> Solid Noise
- Check: Randomize
- Detail: 15
- X size: 16.0
- Y size: 2.0



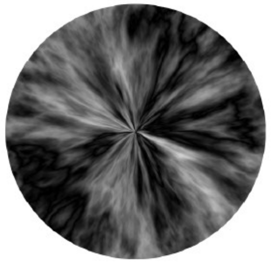
Step6

- Apply polar coordinates to solid noise...
- Filters -> Distorts -> Polar Coordinates
- Circle depth %: 100.00
- Offset Angle: 0.00
- Check: To Polar



Step7

- Move "Polar Coords" layer above pinched layer.
- Set "Polar Coords" layer mode to overlay.

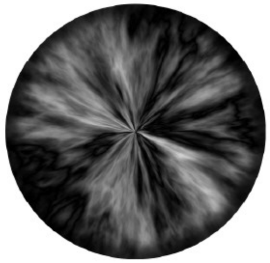


by Benjamin Schram

Step8

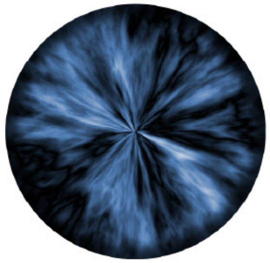
- Select the pinched layer.
- Use the "Fuzzy Select Tool" [u]
- Click in the empty space around the circle.
- Grow the selection...
- Select -> Grow
- Grow Amount: 60 pixels
- Feather the selection...
- Select -> Feather
- Feather: 80 pixels

- Create a new layer
- Name: Edge Ring
- Width/Height: 1024
- Fill Type: Transparency
- Select the "Edge Ring" layer.
- Set foreground color to black.
- Bucket fill selection with black.
- Select None...
- Select -> None
- -or-
- [Shift + Control + a]
- Make "Edge Ring" the top layer.
- Adjust "Edge Ring" layer settings...
- Layer Mode: Multiply
- Opacity: 75.0



Step9

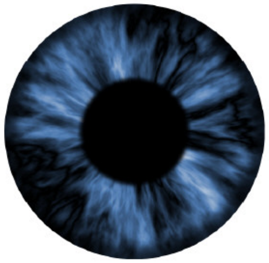
- Create a new layer
- Name: Color
- Width/Height: 1024
- Fill Type: Transparency
- Make "Color" the top layer.
- Pick a foreground color...
- Value and saturation don't matter.
- "0079ff" is a nice blue.
- Bucket fill the "Color" layer with color.
- Adjust "Color" layer settings...
- Mode: Color
- Opacity: Adjust to Taste
- ~40% works well for "0079ff - Blue"



by Benjamin Schram

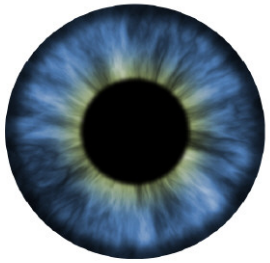
Step10

- Select the "Pinched" layer
- Use the "Fuzzy Select Tool" [u]
- Click in the empty space around the circle.
- Invert the Selection...
- Select -> Invert or
- [Control + i]
- Shrink the selection.
- This determines the size of the pupil.
- Shrink: To Taste (~280 pixels works well)
- Feather selection...
- Select -> Feather
- Feather: 30.00 pixels
- Create a new layer...
- Name: Pupil
- Width/Height: 1024
- Fill Type: Transparency
- Make "Pupil" the top layer.
- Switch to "Pupil" Layer.
- Set foreground color to black.
- Bucket fill the selection.



Step11

- Create a new Layer...
- Name: Color 2
- Width/Height: 1024
- Fill Type: Transparency
- Move "Color 2" layer below "Pupil" layer.
- Select the "Color 2" layer.
- Grow selection (the previous pupil selection)...
- Grow: 60px
- Pick a foreground color...
- Value and saturation don't matter.
- "ffdd00" is a nice gold.
- Bucket fill the selection.
- Select None...
- Select -> None



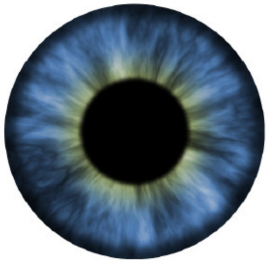
by Benjamin Schram

-or-

- [Shift + Control + a]
- Apply "Gaussian Blur" filter...
- Filters -> Blur -> Gaussian Blur
- Horiz: 150.0 px
- Vert: 150.0 px
- Blur Method: RLE
- Adjust "Color 2" layer settings.
- Mode: Color
- Opacity: To Taste
- ~45% works well for "ffdd00 - gold"

Step12

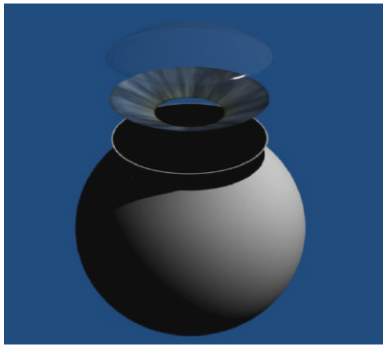
- Tweak "Pinched" layer brightness
- Depending on your colors and noise patterns
- you may want to lighten the "Pinched" layer
- Select "Pinched" layer.
- Adjust brightness/contrast...
- Colors -> Brightness-Contrast
- Brightness : To Taste (+30.0)
- Contrast : To Taste (-30.0)

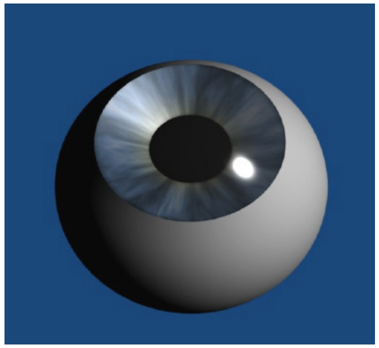


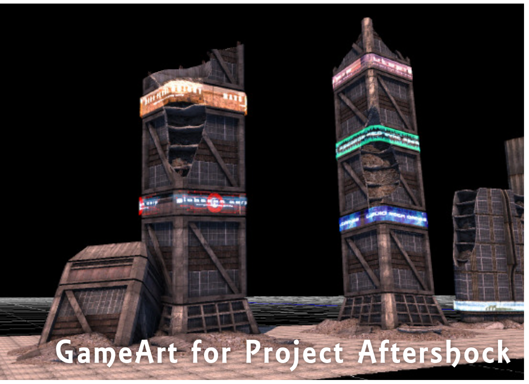
Alright, now head over to the Noob_to_Pro article on “Creating Pixar Looking Eyes” to put this texture to use.

http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Creating_Pixar-looking_eyes_in_Blender

Here are complete and exploded view renders ■







by Yap Chun Fei

Game Art Overview

Producing game art is very different compared to a still render or movie animation art. Game art has the limitation of meeting real-time requirements of the graphic cards. In this article, we will talk about the roles that Blender played in producing game art for our racing game project known as Aftershock.

Hardware Limitations

Before we begin, we have to understand the technology and limitations of our current generation graphic cards. The biggest limitation of a graphic card is the amount of RAM it has. In all games, developers and artist always end up struggling over the ram usage. This limitation brings two important constraints to the artist; polygon count and textures. The artist has to restrain themselves from producing high poly art assets. UV for any art content needs to fully utilize the texture space so as to produce good quality texture detail in a low texture resolution limitation. Adding on to that, the number of textures and polygons in a scene has to be well conserved to avoid hitting the ram limit of lower end cards.

Blender Technology in Games

Blender as a whole serves as both a modeling tool and a game engine. This seemed to come as a winning solution for anybody who wants to make a game.

However, we decided to go for a different model. We use Blender as the modeling tool and Ogre3D with our own built-in extensions with other libraries as our game engine. The reason for this is that we are aiming for a bigger scale game which is graphically intensive. The Blender Game Engine was never designed for such a huge scale project. It does not handle huge scenes well. However, it serves as a very good platform for simple games or prototyping.

On the other hand, the modeling tools of blender work very well for game arts and is in many ways, on par with popular commercial counterparts. This has held true due to the very recent features such as the tangent normal map baking tool to bake high poly to low poly models and the sculpting tool for producing high poly models. In addition to that, the scripting technology allowed us to extend Blender to export blender created content into our game.

Tying Blender with external game engines

To get Blender models and materials out into our Ogre3D counterpart, we used the Ogre exporter provided kindly by the community of Ogre3D. In addition to that, we also wrote our own prefabs exporter script that helped us generate our prefab information from Blender into our game engine specific prefab objects.

Level editor

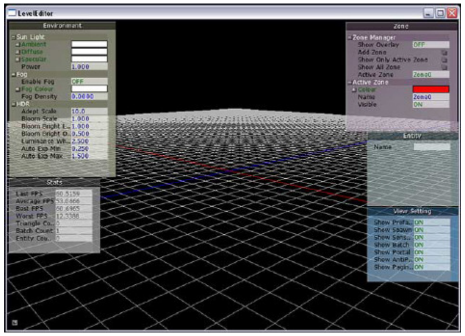
To produce the quality of what is presented in Aftershock, a custom editor had to be created. The reason for this was that it is not possible to build the whole level of such scale in Blender. Adding to that, as we are not using Blender as the rendering engine, what we see in Blender is not what we will see in the game. This posed a huge problem for the artists where iteration is required to produce good art.

by Yap Chun Fei

With a level editor outside of Blender, we eliminated a few problems. Firstly, the artist gets to preview their art content in a WYSIWYG manner. They will not need to go back and forth with the programmer to test and check their art assets. This allowed them to iterate their art, tweak and touch up until they are satisfied with the end result.

The level editor also serves as an important tool for features that are not covered or should not be covered by blender. Two good example for this are terrain editing and grass/bush plotting. In any typical game engine, optimizations are made to keep terrain and grass rendering optimal. Hence they require special data format which is much easier to edit and modify within the game engine. Another good example is the portal zone placement system. The Aftershock level uses portals as a form of optimization to cull off unnecessary meshes that will never get shown in a given area. However, as portal and zone placement is very subjective and relies a lot on how the scene is laid out, this is better done with the level editor where it's much easier to tweak and test.

From the technical aspect of things, the level editor serves as a very good platform to implement game play elements and design level based logic like trigger points and user interactive scene objects which are dependent to the game engine.



Sun Light
Ambient
Diffuse
Specular
Power 1.000
Fog
Enable Fog OFF
Fog Colour
Fog Density 0.0000
HDR
Adept Scale 10.0
Bloom Scale 1.000
Bloom Bright L.. 1.000
Bloom Bright O.. 0.500
Luminance Wh.. 2.500
Auto Exp Min 0.250
Auto Exp Max 1.500

Last FPS 50.5159
Average FPS 53.0666
Best FPS 50.6965
Worst FPS 12.3388
Triangle Co.. 5
Batch Count 1
Entity Cou.. 2

Zone Manager
Show Overlay OFF
Add Zone
Show Only Active Zone
Show All Zone
Active Zone Zone0
Active Zone
Colour
Name Zone0
Visible ON

Name

Show Prefa.. ON
Show Spawn ON
Show Sens.. ON
Show Batch ON
Show Portal ON
Show AntiP.. ON
Show Pagin.. ON

Figure 1: A prototype level editor used in Project Aftershock

Hence, the Level editor served as an important intermediate tool to bridge our Blender art asset with the game.

Modeling

Blender is a very polished polygon modeling tool. The features designed were very useful and helpful in producing low poly art which is very important in any real-time environment. In our project Aftershock, we utilized blender as an object modeling tool. This helped us deal with the details of our individual objects in a practical manner. We are able to control the poly count individually and produce good UV for our objects/prefabs. In a typical scenario, a game level object should never exceed the 5000 poly limit. However, as graphic cards perform faster and faster, this limit will be raised further. Even so, in practice, an artist should always keep their game art's poly to be as low as possible without degrading the art asset into an unidentifiable lump of blob.

Materials and Textures

Materials and textures is what makes an art alive. The fundamental of an art depends on the material which describes the shading, and the texture that defines how the shading works. Blender has a very good material texturing system that works very well with their internal renderer. However, for a high end game which requires custom hardware shaders such as Aftershock, Blender's material system falls short. To solve this problem, we extended the Blender material exporting solution with our own using the custom ID property system of Blender. That allowed us to add additional parameters to the limited selections of Blender materials.

As with any real time applications, there is a limit to the texture usage that we had to observe. Older cards limit any texture size to 2^n (2 to the power of n). This means that texture size must always be in the resolution of 1, 2, 4, 16, 32, 64, 128, 256, 512, 1024 and so on and so forth. Even though non-power of n textures are now technically supported, it is still a better choice to keep them within this limit for optimal rendering.

To alleviate the limited GPU ram as described in the overview, textures can be exported in compressed format known as the DDS/DXT format. This format reduces the memory requirement in the GPU as the textures are stored compressed within the GPU ram itself. However, due to the lossy nature of the format, the texture has some considerably ugly artifacts that might not look good for certain type of textures. Even so, in typical usage cases, as we had found out, the artifacts are negligible and not very obvious. This technique is extensively used in many AAA games on the market today.

Lighting and Shadow

Lighting and shadows play an important role in Project Aftershock giving the city level the overall mood and feel, and shadows give it depth in the game. The lighting and shadow method which we used is split into 2 parts: I)Real time lighting and shadow, and II)Baked ambient occlusion maps.

Traditionally, most games use pre-rendered light maps which are generated either from 3d packages such as Blender or from the level editor itself. The lightmap is then assigned to all models which share the same UV layout and map channel (usually on the 2nd UV map channel on top of the diffuse map channel).

Although generating lightmaps for individual objects and then including them as part of the diffuse texture in a single UV map channel, lightmapping is especially important for level scenes whereby most of the textures used are tiled textures as well as different polygon faces using different materials, making the first UV channel unsuitable for lightmapping where the entire lightmap must fit into the UV map boundaries and this lightmap is shared with different objects, each with its own unique UV map layout, hence the need for a 2nd UV map channel specifically for the lightmap texture.

Pre-rendered lightmap textures usually come in resolutions of 1024x1024 or 2048x2048 (must be in power of 2 for optimum memory usage) for an entire level, depending on the game's target hardware limitations. Graphic cards with greater amount of RAM would be able to use higher resolution lightmap textures. Generating lightmaps with radiosity effects is slow and time consuming where the lighting artist has to wait for the level editor or 3d package to complete generating the lightmap texture before viewing and checking it for lighting artifacts problems (problems such as pixelated lightmaps).

However, as games become more and more detailed and complex, especially when the polygon count has increased a lot, generating lightmaps may not be a viable choice. Newer games such as Assassin's Creed, use real time lighting and shadow with baked ambient occlusion maps. This is because in older games the polygon count for the entire scene is much lower compared to today's levels, an older game may only have less than 50,000 polygons whereby newer games may have 500,000 polygons/level or more. And since we are still limited to 1024 or 2048 resolution lightmaps squeezing 500,000 polygons onto a single lightmap texture produces a lot of lighting artifacts as compared to squeezing 50,000 polygons on a single lightmap texture.

by Yap Chun Fei

by Yap Chun Fei

If the game artist is building an entire game in a 3d package, unwrapping also becomes a major headache and is not the optimum choice for artists. Imagine unwrapping 500,000 polygons for the whole level the first time for the diffuse texture and unwrapping again for the 2nd UV map channel for lighting. This would've taken ages to complete not to mention arranging the UV poly islands on the UV map channel, which would've been a complete nightmare for any level artist. This in return would make any corrective measures slow and cumbersome.

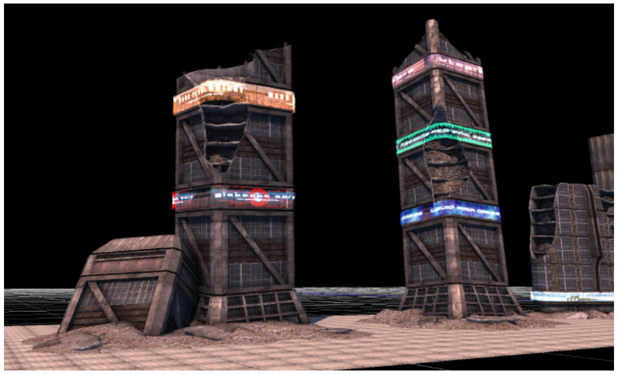


Figure 2: Couple of building blocks in Blender 3D which only has the first map channel textures

Therefore newer games are splitting full scene lighting/shadow and soft shadows(ambient occlusion) separately. For the Project Aftershock game, each building and track has its own baked ambient occlusion lightmap, whereby scene lighting and shadow is done real time in the level editor which allows the artist to iterate and correct any problems very quickly. Here is how we generated the ambient occlusion maps in blender:

As we can see due to the lack of soft shadows around the building corners it currently looks flat. First create a new UV map channel for the textured building model for the lightmap texture under editing panel.

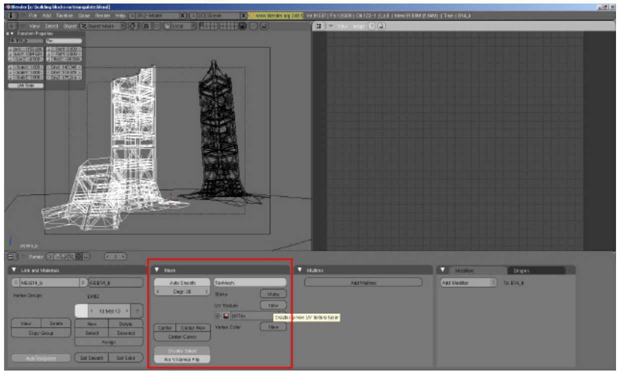
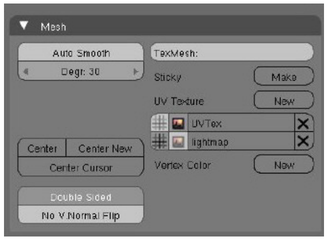


Figure 3: Creating a new UV map channel for the ambient occlusion lightmap texture .

Press New to create a new UV texture layer and rename it to “lightmap”.



Make sure while still in the lightmap texture channel press the [TAB] key to go into edit mode. The next step is to triangulate all the faces. Still in edit mode select all faces to be triangulated by pressing [a] key.

This is important because to bake lightmaps/ambient occlusion correctly, Blender will not be able to tell the shape of a polygon face orientation which may cause lighting artifacts problems.

by Yap Chun Fei

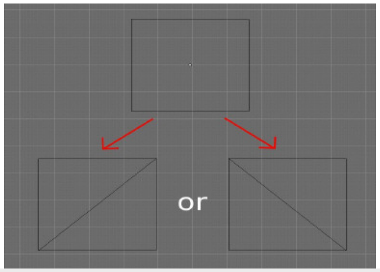
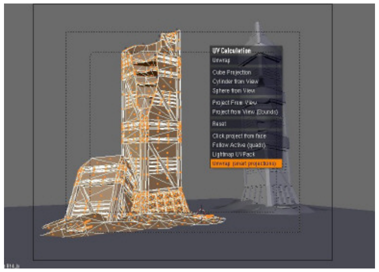


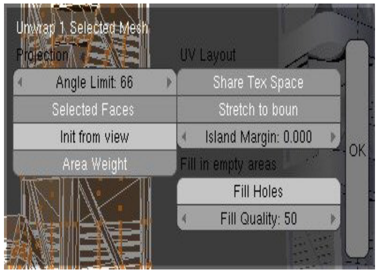
Figure 4: Without triangulation blender will not be able to tell the correct face properly causing lightmap baking artifacts where shadows are cast on faces where they're not supposed to.

Triangulate the selected faces by pressing [CTRL-T]keys. Warning: It is highly recommended that the artist is thoroughly satisfied with the initial object textures before triangulation as pressing the "Join Triangles" key under Mesh tools tab will mess up the initial building's UV map should the artist decide to redo the 1st map channel's textures.

Once the faces are triangulated we need to unwrap them. Press the [u] key to show the unwrapping list. Select "Unwrap (smart projections)" to unwrap. This method of unwrapping is selected because UV island distribution is based on the actual polygon size compared to using "Lightmap UV pack".

After selecting "Unwrap (smart projections)" a menu will appear. Select "Fill Holes" set fill quality to 100, "Selected Faces" , "Area Weight", and set island margin




Unwrap 1 Selected Mesh
Projection
Angle Limit: 66
Selected Faces
Init from view
Area Weight
UV Layout
Share Tex Space
Stretch to boun
Island Margin: 0.000
Fill in empty areas
Fill Holes
Fill Quality: 50
OK

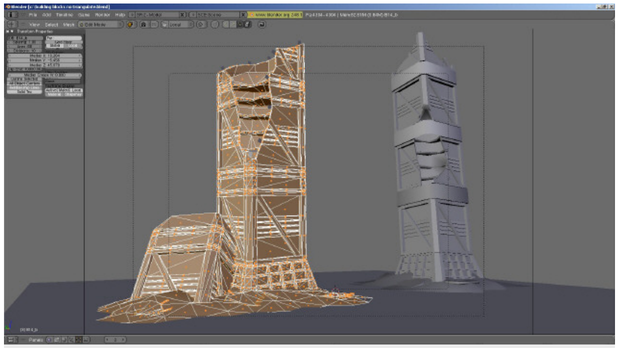


Figure 5: Triangulated faces.

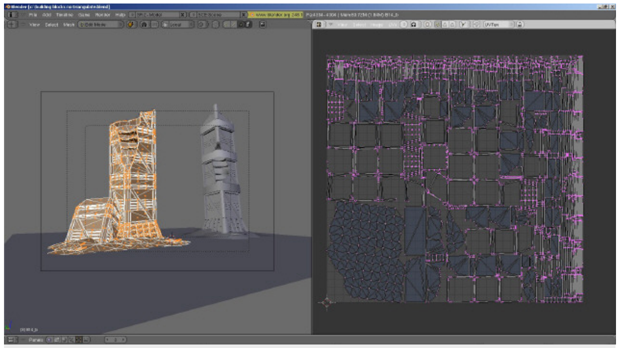
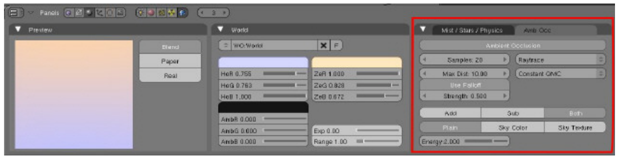


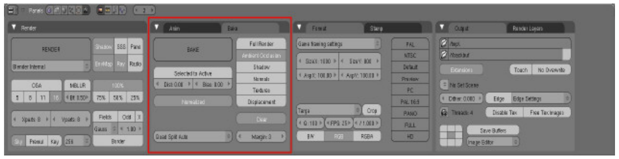
Figure 6: Unwrapped building using Unwrap (Smart Projections)

by Yap Chun Fei

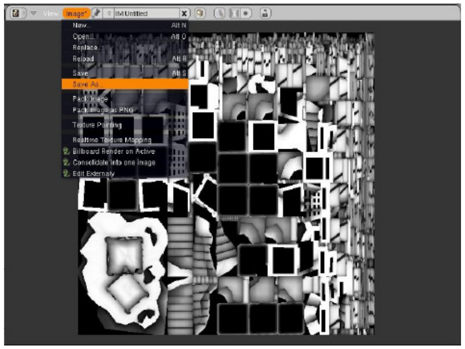
Still under the UV/Image Editor window ,go to Image>>New to create a new texture image for the light-map. Now we are going to set our ambient occlusion settings. Go to “World buttons” panel and enable Ambient Occlusion. Here the artist can adjust the ambient occlusion settings to fit their model.



Go to Scene (F10) panel to begin rendering the ambient occlusion lightmap texture. Under “Bake” tab, select Ambient Occlusion and Normalized and click on BAKE to begin rendering.



Once the ambient occlusion lightmap render is complete we need to save the new image file within the UV/Image Editor window.



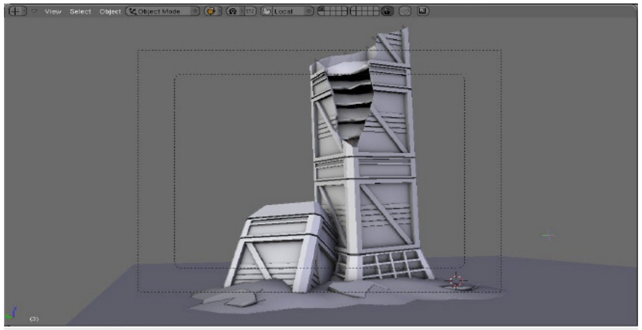
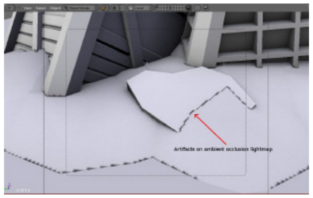
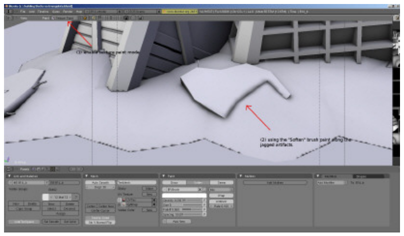


Figure 7: This is how the building model looks like with ambient occlusion map

After saving the new ambient occlusion map it is now time to clean up any rendering artifacts.



To fix the artifacts, go to Texture Paint Mode and using the “Soften” brush, paint along the jagged edges to blur the problem areas. This produces a much softer look and feel for the soft shadows. Once completed, save the corrected ambient occlusion lightmap texture.



And finally, this is how the building model looks like in the level editor with both diffuse map channel and light-map texture channel combined together.

by Yap Chun Fei

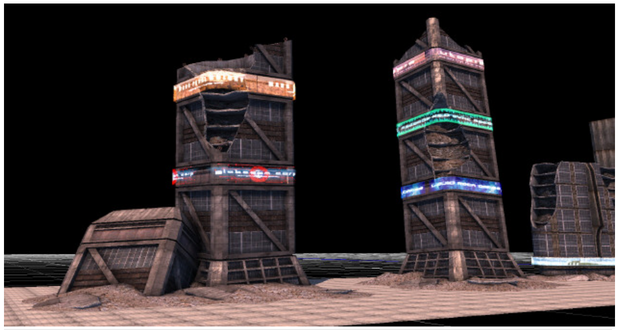


Figure 8: Building models with ambient occlusion maps.

Lighting and shadows are calculated real time within the custom built level editor.

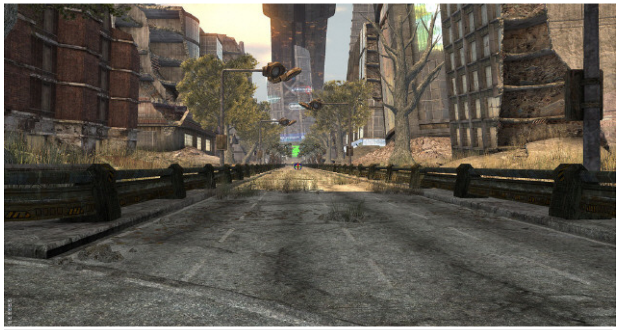
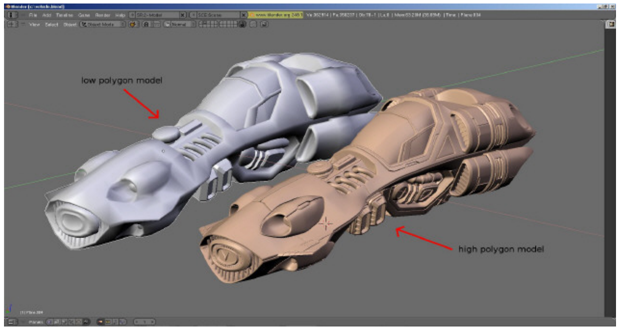


Figure 9: Final building models with ambient occlusion lightmap and real time lighting and shadow.

BAKING NORMAL MAPS AND AMBIENT OCCLUSION MAPS USING A TEMPORARY "CAGE" MODEL

Modelling the vehicle craft for Project Aftershock requires both high and low polygon models where by the high poly models provide the extra details through normal maps and ambient occlusion maps. However we will write an additional tip for generating proper normals and ambient occlusion maps.

First of we will require both a high poly model and a low poly model. Whether the high polygon model is built first and then optimized to a lower polygon version or vice versa is entirely up to the artist. For this craft's pic, the low polygon model is approximately 8000 polygons.



After that we will then make sure the high and low polygon version are exactly in the same position to bake the normals and ambient occlusion maps from the high polygon model to the low polygon model. After that we will need to unwrap the vehicle model and create a new image to bake to.

by Yap Chun Fei

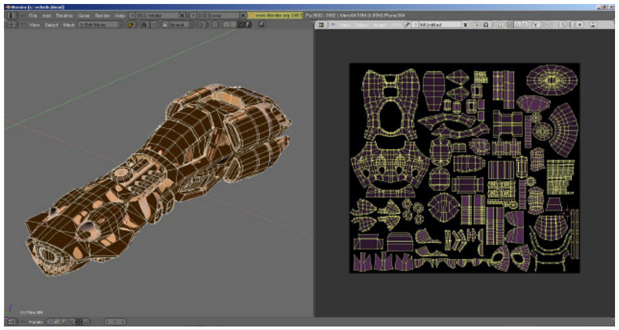
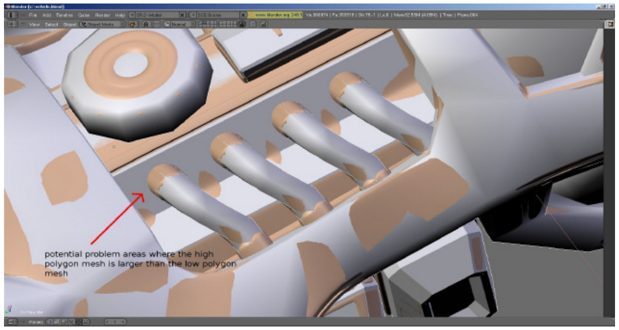


Figure 10: Low and High poly models together with unwrapped low polygon model

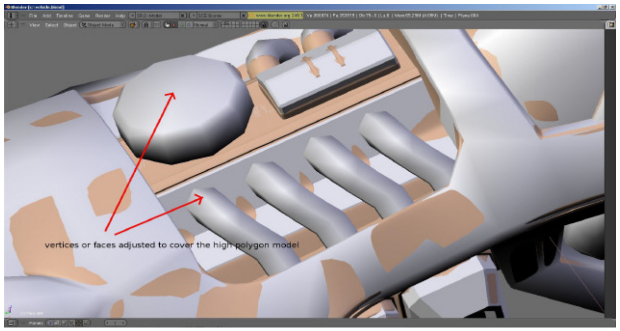
The next step is to create a copy of the low polygon model. The reason for doing so is that the low polygon model which has the same position with the high polygon model will act as a "cage" similar to the projection modifier in 3ds max.

At this point of time we have 2 low polygon models(1 temporary "cage" model and 1 to be used in game) and 1 high polygon model.


potential problem areas where the high polygon mesh is larger than the low polygon mesh

This cage is particularly useful in modifying only certain parts of the low polygon mesh to fit the high polygon mesh since Blender adjusts the baking distance on the overall low polygon model.

Next step is to readjust the vertices or faces of the low polygon model to cover as much of the high polygon model as possible.



Once this step is done we can then proceed to baking normals and ambient occlusion texture maps for our low polygon cage model. To do that, select both the high polygon model and low polygon model (with the low polygon model as the active object), go to SCENE (F10)=>Bake (Normals or Ambient Occlusion) with "Selected to Active" option turned on. Once we have completed generating our normals and ambient occlusion maps. Re assign them to the first low poly model and delete the temporary "cage" model and that's it! 🟥

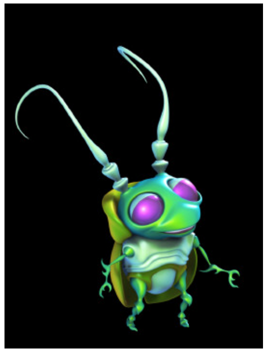
by Benjamin Schram

Introduction

The series of images that I created for "Cricket and Friends See a Comet!" represents my first effort at 3d character creation. During the last year of my Fine Arts program at the University of Wisconsin - Eau Claire, a professor from the Astronomy/Physics department asked me if I would be interested in making some characters for a children's show at the university planetarium. The show's visual elements would include five woodland animals and very minimal props. Since the characters would be on pure black backgrounds, to superimpose cleanly on the dark planetarium dome, there was no need for additional scenery. It was a great opportunity for me to really dig into creating expressive and flexible characters from scratch!

Since this is more of a showcase than "how-to", let's just look through the characters and talk about aspects of their design...

CRICKET



Cricket presented a number of unique design challenges. She is by far the smallest of the characters and, unlike the rest (of them), has a rigid body with no fur or feathers. The need for the characters to have emotive capacity and for the image to have a black background both contribute to Cricket differing more than any of the other characters from her natural world family in color and body structure.

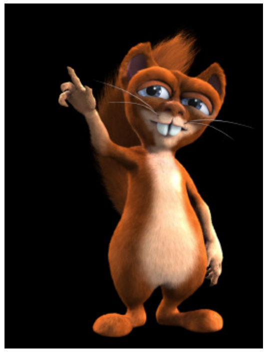
Color

Real crickets are black, however, as the show is to be projected floating on a planetarium dome (black sky), black equals transparent. Green with iridescent analogous hues (via node based, normal angle color-shifting) seems to give her a strong presence without flattening out. For her eyes I used a complimentary color to add further visual weight.

Design

In order to be recognizable as a cricket, the character needs very thin limbs, which combined with a rigid body mass creates a very dull character profile. To combat this, I exaggerated the "spiky" aspects of her limbs and really maxed-out her antennas. These few adjustments to the basic body shape, emphasize the angle and direction of arms, legs, and head giving each pose a much more recognizable profile.

SQUIRREL

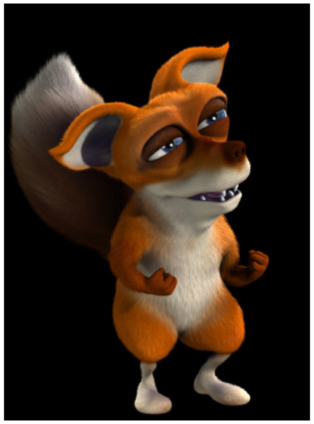


Squirrel was a lot of fun. Her lanky and flexible body made her easy to form into interesting poses. One problem that I encountered was scale discrepancy between characters. Squirrels are clearly smaller than foxes and bears, but since the size of the final projected images is not very large, she could not actually be much smaller. To make her feel smaller I made her eyes very large relative to the size of her head.

by Benjamin Schram

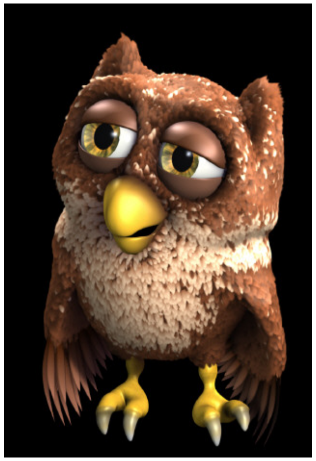
This, combined with long, thin limbs, turned out to be very effective at making her seem less massive without actually making the character smaller.

FOX



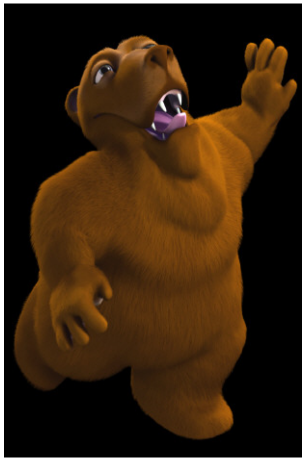
Fox has the most varied color patterns. In order to get precise color placement and smooth blending across UV seams, I did most of the painting directly on the model rather than in The Gimp or the UV/Image window. She also (along with Bear) has a more complete mouth than the other characters including lips, a set of teeth, a curve deformed tongue and fully modeled interior.

OWL



By far the most challenging aspect of Owl was the feathers. I ended up using four particle systems for all the feathers except the longer wing feathers. I used weight painted vertex groups to define particle densities for separate white and brown particle systems. This worked very well for creating Owl's "speckled" areas and color gradients. For more control in troublesome areas, I needed to create a second set of brown and white particle systems to manually place individual feathers around the eyes, "ears," and beak. The wing feathers are individually placed meshes. Each of the two sets of wing feathers has a mesh deform modifier to allow grasping and gesturing with the feathers.

BEAR



Bear presented the opposite problem from squirrel. He needs to appear massive without being so. To make him feel as large as possible I gave him small eyes relative to his head size, and very massive limbs and body. Having such thick arms, legs, and body made him a challenge to rig and pose. The mesh deform modifier was a lifesaver!

The inspiration for the design of his head came from a sketch sheet at www.creaturebox.com. I highly recommend that anyone looking for some good character inspiration check out the site and even buy the CreatureBox Volume One Sketchbook.

by Benjamin Schram

The images for the show are now all complete and passed on to the planetarium. I have learned a LOT over the course of the project and the final content has been very positively received by the cooperating faculty members and script author. I have also discussed open licensing with the others involved and think that there would be no resistance to sharing the show with any other interested planetariums under a creative commons license.

This project benefited a great deal from the Peach Project. The improvements to the fur/particle system and the creation of the mesh deform modifier made the fur and rigging far easier and more powerful. The accompanying blog posts, documentation, and source files made learning how to implement all the features possible! Thank you to the Blender developers for creating such an excellent program. After working with XSI in the university's 3d classes, it was a relief to switch to Blender's (non-modal) interface. And now that I have graduated, I would no longer even have access to 3d software if not for Blender! 🟥

Benjamin Schram

Benjamin Schram is an artist from Wisconsin - USA. He graduated in 2008 with a BFA in drawing from the University of Wisconsin - Eau Claire. Currently he works freelance, doing 3d and Illustration.

CrowLine Studio
Website: www.crowlinestudio.com
e-mail: www.crowlinestudio@gmail.com

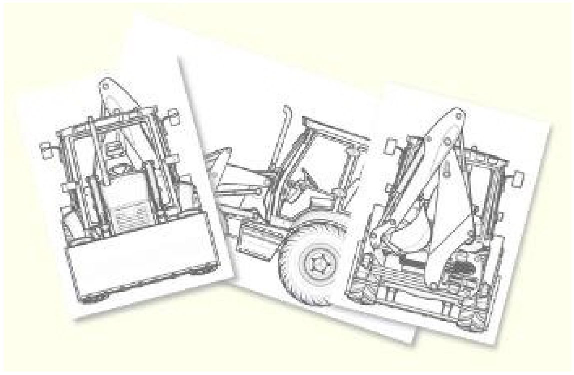
By Robert Burke

Introduction

I have been working on this model for some time now and rather than repeating the usual step by step tutorials, I have decided instead to write a brief article. The article will detail the process and a few of the techniques used to get from the first idea to construction of a complex mechanical model that could be animated, to the finished excavator.

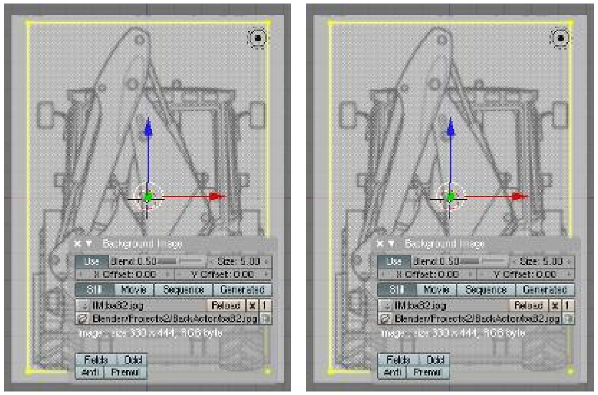
As with any project the first stage is planning, and as this was a learning exercise to get to grips with Blenders animation tools, I had to choose a machine that had a lot of moving parts. The excavator fit the bill perfectly.

Now knowing what I was going to model, it was time to go off and find reference materials. A set of plans would be useful, but unfortunately none were available on the usual plan repositories. However, the manufacturer did have some reasonable side, front and rear views in their pdf literature. They didn't exactly scale through from one view to the next, but a little adjustment in Gimp got them near enough to be used.



Besides the plans, Google was useful for finding photographs, and whilst passing a near by construction site, an excavator was parked alongside the fence allowing me to have a close up look at the machinery.

Setting up plans in blender is quite a simple process, I usually work from three 3D views and toggle the active view to full screen with Ctrl-down arrow. As I only had a front, back and side view of the excavator, these were set up.



To set up the views, in the 3D view header click View>Background Image and from the background Image window load the front view. To help align the other views drag a plane so the edges touch the extremities of the excavator front view. In the next 3D window load the back image and then use the X Offset, Y Offset and the Size controls in the Background Image window to scale and align the image to the edges of the plane. For the front view you will need to rotate the plane 90 degrees on the Z Axis and

by Robert Burke

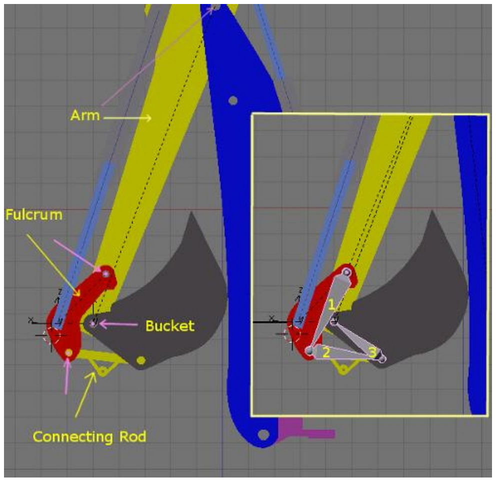
align the image only to the top and bottom edges.

Before I set about modelling the excavator, I thought it would be useful to make sure I could get it to animate. To do this, I simply traced the arms and buckets using Bezier Curves and used these as simple 2D cut-outs to help position the armatures.

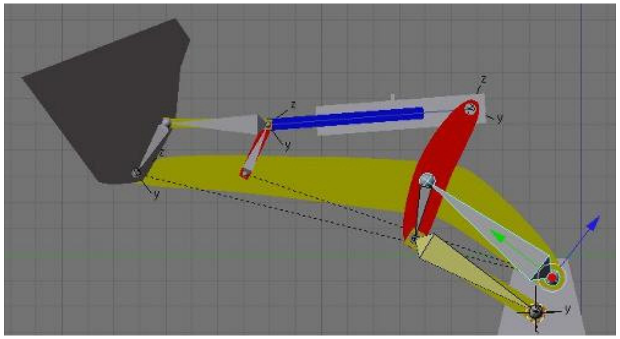
The pistons and cylinders used simple Track To constraints with their ends parented to the respective component, in this case the cylinder to the arm and the piston to the fulcrum. The process is described really well in the Blender 2.3 Guide and is still relevant to the current versions of blender.

It should be stressed that pivot and rotation points must be accurately set and aligned using Blenders Snap tools.

The bucket movement was a little more complicated than a straight armature chain, requiring both ends of 3 bones to be set to fixed locations. This was achieved by adding an Armature Object at the pivot point of the fulcrum and parenting it to the arm and snapping the other end of the bone to the pivot point of the connecting rod. Another bone was extruded from this point and snapped to the interconnection between the Connecting Rod and Bucket. Then a third bone was extruded and snapped to the pivot point of the bucket. An empty was also added to the pivot point of the bucket and parented to the arm, so the empty would move in relation to the arm's movement.



The Fulcrum was parented to Bone 1, the Connecting Rod parented to Bone 2 and the Bucket to Bone 3. The trick to getting this type of mechanism to work was adding an IK Constraint to Bone 3 with the target set to the empty I had placed on the Bucket pivot point, with Use Tail selected in the IK Constraint.



The two unused axes of the bones were disabled in the Armature Bones panel, allowing the bones to only rotate on the one axis. To animate the bucket movement now only required Bone 1 to be rotated in Pose mode and all the other components would follow.

The front mechanism used a similar setup on both the front bucket rotation and the top cylinder fulcrum.

Satisfied that the process of animating this machine could be accomplished, it was time to start modelling.

MAKING OF: The Process of Modelling and Rigging a Mechanical Model

by Robert Burke

If you have read through the Precision Modelling tutorials, you should be familiar with all the tools and techniques needed to create any mechanical models.

In fact its even simpler to create a model for animation, as it only needs to be a representation of the real thing and not a CAD perfect 3D reproduction.

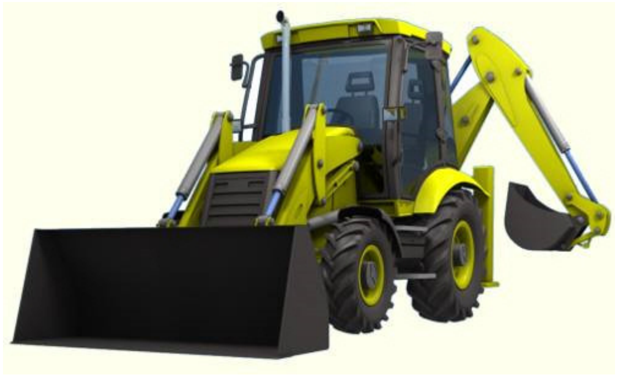
The process is therefore simply a matter of tracing areas of the model from the background image in one view and then pulling the vertices to align with the corresponding point in the perpendicular view.

The job is made even easier because the Tractor unit and front mechanism is symmetric between the left and right sides, so only one side needs to be modelled with the other being created using a Mirror Modifier.

On complex models, working component by component can be a lot less daunting than trying to build the whole model as one mesh.

Once comfortable with the process described in the Precision Modeling tutorials, it should become quite straightforward to build large complex models from numerous components. However being able to animate those models means you need to be fairly strict in the way you name each component and in the structure of how they are parented to each other. The Mirror Modifier was a useful short cut for modeling, but can cause some problems further down the line on this project, so I applied the modifier once the modeling was finished.

With the basic model completed and all the parts parented to either the armature systems or the main body, it was time to start adding details.

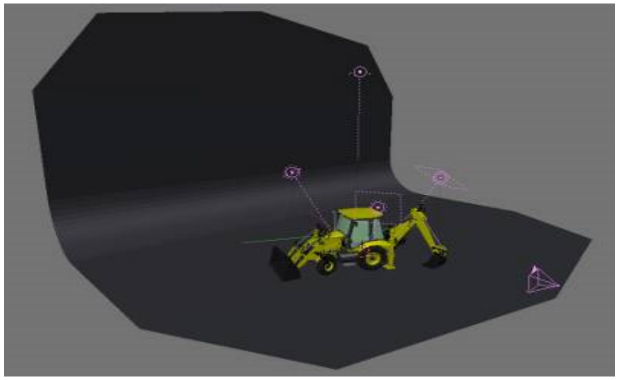


by Robert Burke

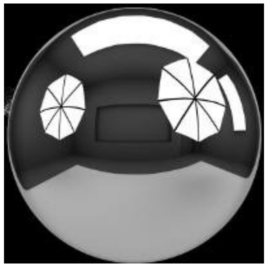
Moving hydraulic components need hydraulic fluid to be pumped through them, so this necessitated pipes and hoses. Pipes aren't a problem because they are static items in relation to the components they are attached to, but hoses need to flex and move with the animation.

I experimented with a number of methods to achieve this, but found you could obtain a very realistic movement using just curves, hooks and empties. What's more, you can set the end tangency of the hose so it doesn't rotate at the end point. I will write the method up as a mini tutorial sometime in the future.

With a few more details added, it was time to see how this model looked in a render. Basic materials were added for the paintwork and a highly reflective material added to the hydraulic piston to simulate the ground chrome material. The model was then positioned on a plane which was curved at the back to simulate a studio back drop. Two area lights were added together with a light behind and above the model relative to the camera.



In order for a reflective material to look good it must have something to reflect.



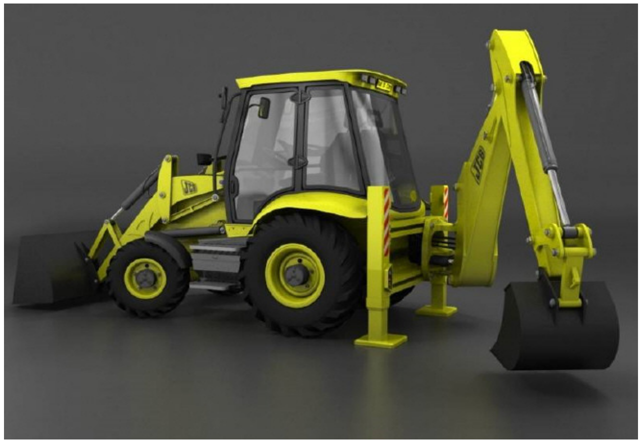
A basic reflection map was created by placing a UV sphere inside a box together with a few props to simulate a studio flash brolly.

The UV sphere was set to the maximum reflection and an image renderer of the sphere. This was then used as an angle map in Blender's world settings

The end result was a fairly convincing image of an excavator, though, given time, the image could be improved considerably.

This isn't the end of the excavator project; I now need to UV unwrap the model and add some realistic textures, then set the model in a construction scene. All that, however, is for a future date and I will post further information here when it is completed.

I hope you have found this run-through of the project useful ■



By Sozap

Introduction

And now let's have a trip deep into the sun and see how it's made :

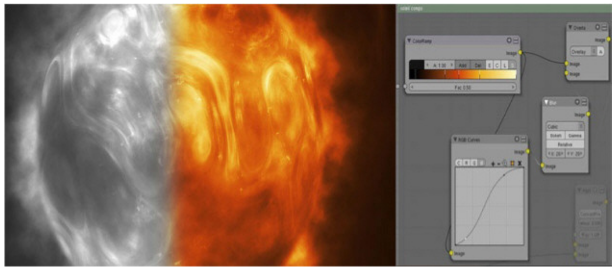
First we have very nice photo references (that you can see on the animatic).

With that, half of the job was already done :°D , because sometimes you spend more time finding what you want to do than making it. In this case it was already there. With that done, the next thing to do was to sit down and do nothing, except thinking:

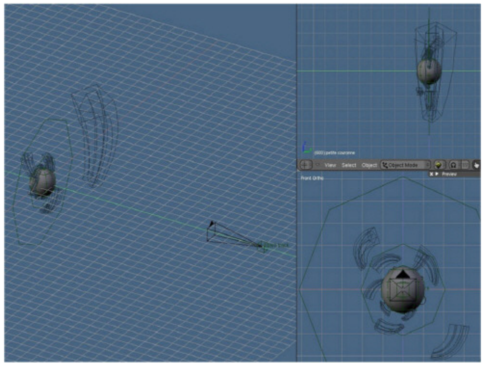
Before starting a project, it's always a good thing to list every possibility that you might have to consider to get the effect you want. Think about the pros and the cons, how long it'll take to accomplish... When this part is also done, you still have nothing in your blender, but you might be more advanced than the guy who started blending from the beginning.

From that, I decided a few things:

First, I will do the color in the compositing, so I can easily tweak colors at the last moment, and also I don't have to worry about colors in my texturing.



Also, as a star is a very big object, and in the animation we don't turn around, instead of making a complete 3D object, I can make it 2D/3D or a kind of a matte painting. That allows me to do even more cheating.



Here you can see the final object in the 3D view.

Finally, I decided to use procedural textures as much as I could, so everything could be appended in blender, and it's non-destructive. At any time I can change any value.

Then I started to look at the references images, to think how I will reproduce the various elements that form the picture of the sun.

So now after that reflection phase, I started a testing phase, which turned out to be the final sun two or three days later.

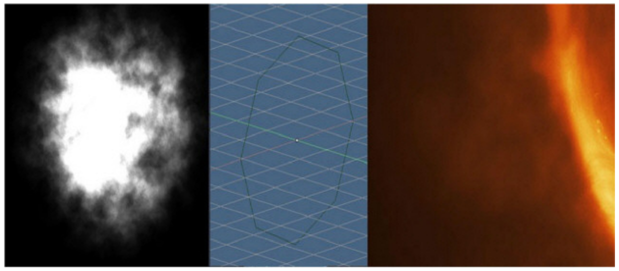
by Sozap

Here are the different elements that I've made :

For the "cloudy flames" around the sphere, instead of using particles, I preferred using a simple flat mesh with a bunch of procedurals.

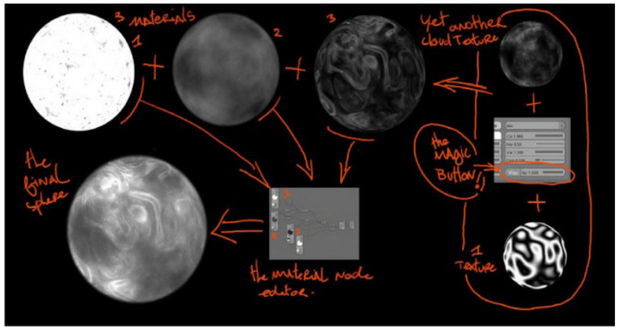
Here is a link to a tutorial that covers this kind of effects, my setup is not exactly the same, I don't use sticky, but the spirit of the thing is here :

http://stblender.iindigo3d.com/tutorials_advancedshockwave.html



For the sphere, I mix various materials in the node editor that each have one function :

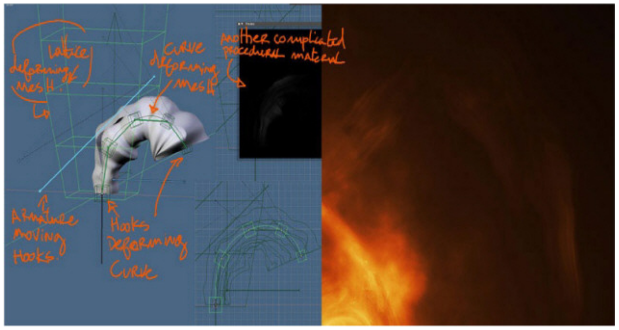
1st material makes the white spots,



The 2nd makes a global gradient on the surface, and has soft edges, that allows this material to fade with the previous « cloudy flames » and the 3rd makes the psychedelics waves on the surface.

For the flames (the eruptions ones), I used a combo of modifiers on a mesh:

lattice + subsurf + displace + curve, all this is parented to an armature to animate it easily. Then I make a group of it, make an animation of the flame (that is really slow, so you won't notice it in the clip) and then using dupligroup + timeoffset, I put other flames around the sphere.

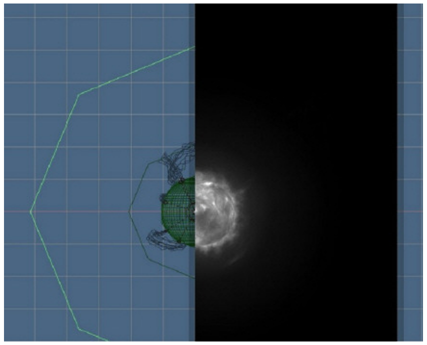


Here is a link to a tutorial that explains the armature + hook + curve trick :

http://wiki.blender.org/index.php/BSoD/Introduction_to_Rigging/The_Bones-on-Curve_Spine animation

BSoD This introduction to rigging is a very good starting point if you want to understand and make useful rigs, and there are other interesting subjects in BSoD as well: animation, lighting, modeling, materials, principles of animation...

by Sozap



At that point everything was ok; there seemed to be just a little thing missing, so I added this second big circle, which has a sphere blend texture, that makes this lighting/ halo effect. And it fades nicely with the "cloudy-flames."

Now it's time to animate:

Every texture was mapped using coordinates of empty objects. Here is a very good tutorial that explains this, and if you want to learn more about procedural texturing it's a very good one:

http://wiki.blender.org/index.php/Tutorials/Textures/Wood

So to animate the textures, I've moved the empties. In fact, I've concluded that in this case, the slower it moves the better effect it gives, so maybe if you don't pay attention you won't see the texture moving.

About render times :

As I'm using a shadeless material, and there is no raytracing, or very much geometry, the render time is nice. For instance, rendering this frame (768x432) on my core 2 duo 6600 took less than 12 seconds. When I add textures and effects, I always check the rendering time; ideally, I want my frames to render between 10s and 1mn. For those who still believe that a good result equals long rendering time, I hope that I've demonstrated that it's not always the case.

To conclude:

I hope you've enjoyed reading all this, and you've also learned something; even if I didn't show you exactly how to do it, you can see that a complex object is always made of simpler parts. It's always a good thing to mix various little effects, instead of having just one big one.

That's all, thank you for your attention, and keep blending!! 🟥

Sozap

MAKING OF: A Sneak Peek at GAME THEORY (Short)

By Giancarlo Ng

Introduction

This Fall 2009 (hopefully) will see the Internet premiere of a new 2 minute animated CG student short called "Game Theory", from first-time collaborators William Oglesby and myself. The film, to be made entirely with Blender, is a sci-fi fantasy that focuses on the climax and conclusion of a tense game of Chess between a young gifted human Chess Grandmaster who matches wits against a fictional Chess Playing Super-Computer.

Story and Concept

"Game Theory" is inspired first and foremost by the famous Man versus Machine themed Chess matches of the 1990's and borrows elements and sub-themes from these encounters. In particular, it borrows the image of the dapperly-dressed smart man pitted in piece-pushing combat with a dark, inanimate object constructed it seems, entirely of nothing but pure cold logic wrapped in steel. While this student project is angled as an experiment in first-time collaborative work using the Blender software, the project also aspires to explore the condition of the proverbial Overachieving Man in the form of the film's protagonist.

PROJECT FORMATION

Writing and Inspiration

The actual story writing for "Game Theory" started in June of 2008. I had always been interested in "Man against Machine" themed projects and found inspiration in particular with the matches between Gary Kasparov and IBM's Computers. I also drew upon observations of other "Kasparov-type" achievers - particularly honor students and champion athletes - to help flesh out a story. In its final form, the vision is that the story will be more like a single moment in time at the climax of a critical moment of the said achiever's chosen activity of excellence.

The story was completed around July of 2008, alongside selection of audio samples for the student project. The selected audio then influenced the writing of an actual script which was completed between August to November of 2008 during breaks in my other animation project "The Surprise Attack".

With the realization that one artist would have trouble achieving the entire scale of this project alone, William Oglesby was contacted to join "Game Theory", which then became a 2-man student learning project in January of 2009. The idea was that we were going to learn from each other, and learn new things together while trying to fulfill particular demands of the film.

Along this time, the project had taken on numerous "smaller inspirations", mostly from science-fiction films and new ideas which William had passed on to me. These new inspirations led to re-writes of the script, which began alongside early Concept Art which I started to create as things were finalized. The re-writing period finally concluded with a "near final" in March 2009. I say "near final" because nothing in this type of field is ever final anyway until you have a copy that has finished the Editing process. Storyboards were completed in March 2009 based on early builds and visualizations of objects as they materialized from their approved descriptions in the Story and Script.

by Giancarlo Ng

Project Team Members

I had been working on building a 400-part transformable robot actor in Blender at the time when the inspiration came to write "Game Theory". I realized while working on the convertible robot, that projects of better scale and quality will not be possible in a time-bound manner while working alone. This led me to seek out at least one more teammate with which to complete the vision for the student project "Game Theory" as well as to pursue further learning in Blender mesh cage rigging techniques, design, modeling, animation, cinematography and direction while working on a "small scale narrative".

For this project, I perform the following roles:

- Director and Animation
- Writer (Story and Script)
- Storyboard Artist
- Concept Artist
- Modeling, Texturing, and Rigging (Human Actor, Other Selected Elements)
- Sound Editing
- Film Editing and Post Production

William Oglesby had previously been working on a multi-layered rendition of a high-detailed London-style clock tower using Blender before he joined production of "Game Theory". William is also an experienced modeler, animator, computer artist, and video game programmer and team leader with particular knowledge in graphics theory, 3D physics, and game engines.

For this project, William fulfills the following roles:

- Visual and Special Effects
- Physics Simulation
- Modeling, Texturing (Sets, Props, "Monolith" Super-Computer)
- Lighting and Other Optical Effects
- Compositor Nodes and Post Production Effects

PROJECT OBJECTIVES

I want to make it clear that while we follow a very organized approach to making this project, this is pretty much a student project. It is born mostly from my personal realization that better results for the entertainment of others and learning about Blender will not be attained without collaborators. Working alone has a tendency of slowing down the learning process, since it seems every time a new project is started, one has to start all over for every element that has not been pre-created. This is time-consuming, energy-consuming, and may result in compromises where a lone artist begins to "rush" elements in an attempt to learn. This is always at the expense of a project's finished look.

One key objective, therefore, was to accelerate and normalize the Blender learning process, by way of collaborative group work. Aside from reducing the steep climb to learning, collaborative groups increase the number of "source imaginations" and inputs exponentially, and the extra hands at work promise increased depth and higher results for each element in a project.

In particular, William and I intend to complete "Game Theory" hoping to learn about collaborative work-flows, organized production directory structures, use of Linked Groups, advanced Compositor effects, Mesh Cage rigging and animation, basic organic facial acting, working with physics in a short film, UV mapping, Sub-Surface Scattering in relation to compositor effects,

by Giancarlo Ng

as well as the “soft skills” in writing, storyboarding, screen testing, cinematography, and basic storytelling.

PRE-PRODUCTION

Music and Audio

In many schools like Animation Mentor, students begin learning about the nuances of film animation by taking their music and audio from films and other professional samples and re-appropriating them for learning purposes. The same approach is taken here. In line with lessons learned from my very first project, Music and Audio were selected prior to writing the Script, but after a basic Story and Concept had been decided upon. In a step up from my previous work, the Audio Track for “Game Theory” features some heavy editing - a title theme culled from a video game, and a main narrative audio track re-using an edited version of a background music track combined with free sound effects and spoken audio from a feature film that has been re-purposed for this student project.

If the union of Message and Theme create the Story, then the union of Story, Imagination, and Temporary Audio create the Tone, from which a more solid Story and Script can be written.

All sources have been noted and will be included in credits.

The Human Actor Protagonist

The story in “Game Theory” is one of some slightly ridiculous, but somewhat serious gravity, and in that sense, I made a decision as early as July 2008, that the actor had to be of “semi-realistic” quality. This goes against the grain of normal student projects where Stylized Humans are more often used. The story called for an actor that was young, appeared smart or talented, and had to dress sharply in formal wear.

Translating these qualities required learning about modeling, rigging, UV Mapping, Sub-Surface Scattering, and Shape Keys. An early proto-type for the actor can be spotted in “The Surprise Attack” as the cowering human standing on the helipad.

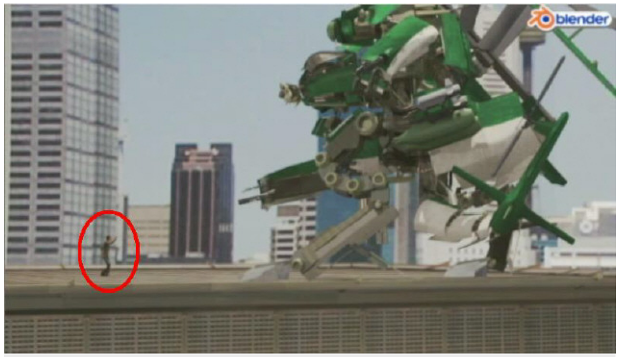


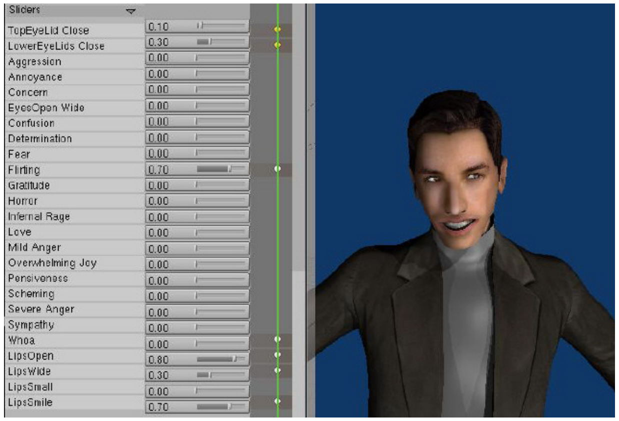
Figure 1: The Human Actor in his debut role.

Results from that initial test led to a re-working of the overly-complex actor rig in January 2009 to fit the actor into a much simpler rig that relied on a Mesh Cage Deform Modifier for all movement except fingers, jaw, and facial shape keys.

We are, of course, under no illusions that we can create a photo-realistic human actor from our current skill level in Blender. Hence there was a conscious decision to mold the actor in the “Image” of a human being rather than a literal pain-staking Debevac-style replication of human “Features”. The model for the human actor was based loosely on photographs of a real-life actor although simplifications were made to his facial geometry.

by Giancarlo Ng

Simplifications were also made to the hair, which is just a single mesh, as well as to the Shape Keys, which control blocked off "whole expressions" where each slider, rather than controlling one part of the face, instead commands an entire facial expression from the nose up. Separate Shape Key Sliders control the lips to help form different mouth shapes and another set of Shape Keys control eyelids. A Jaw Bone construct controls Jaw movement. Currently, there are no plans to use multiple image maps for veins. The sub-surface scattering settings are simply applied over a texture map of the skin, which already includes painted-on hair, eyebrows, lips, and finger nails.



Sliders	
TopEyeLid Close	0.10
LowerEyeLids Close	0.30
Aggression	0.00
Annoyance	0.00
Concern	0.00
EyesOpen Wide	0.00
Confusion	0.00
Determination	0.00
Fear	0.00
Flirting	0.70
Gratitude	0.00
Horror	0.00
Infernal Rage	0.00
Love	0.00
Mild Anger	0.00
Overwhelming Joy	0.00
Pensiveness	0.00
Scheming	0.00
Severe Anger	0.00
Sympathy	0.00
Whoa	0.00
LipsOpen	0.80
LipsWide	0.30
LipsSmall	0.00
LipsSmile	0.70

Figure 2: The "semi-realistic-but-simplified" approach.

This takes us to probably the most important aspect of Pre-Production and the creation of this actor: Feedback and Testing. Simplification of this actor will be for naught if the compromise goes too far in watering down the intended audience perception and effect of the Story. Since January 2009 and ongoing into the present, the Actor for "Game Theory" has gone through a feedback cycle with focus groups (mostly women), who gave notes about his general appearance, which are then noted against the intent of the Story. Eventually the Actor's rig and Shape Keys will be tested in a sample acting reel, and a screen test process will be completed with the set and props developed by William to determine if it "gels".

The feedback cycle also has "soft" elements. During this process we got some feedback determining what might pass for "smart casual with a bit of sci-fi" with the potential audience, and then compared it to early concepts and Story direction. We also learned that "Taupe" was a fancy word in fashion for "Dark Grey Brown".

The Human Actor for "Game Theory" also figures as our first project with an actor rigged using the relatively new Mesh Deform system of Blender. Mesh Deform was developed based on a Disney-Pixar solution for Harmonious Coordinates, which itself, I believe, was used as the rigging solution for Rhino in the movie "Bolt". It is in moments such as these where one has to note the great advances that an open source application like Blender can make in comparison to developments in film and motion picture science.

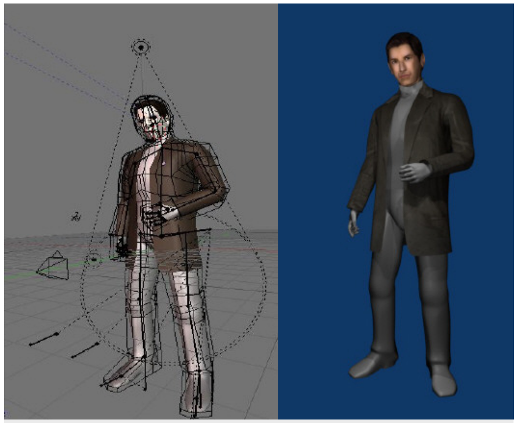


Figure 3: Mesh Deform Cage in Action.

by Giancarlo Ng

Originally the Actor in his previous state featured many control bones, as many as 8 to 10 in a shoulder alone with many solver constraints. The goal behind using a Mesh Cage was to simplify the process by weight painting a very low poly mesh and then allowing Blender's Mesh Cage Modifier to intermediate the changes to a high-poly Actor Model.

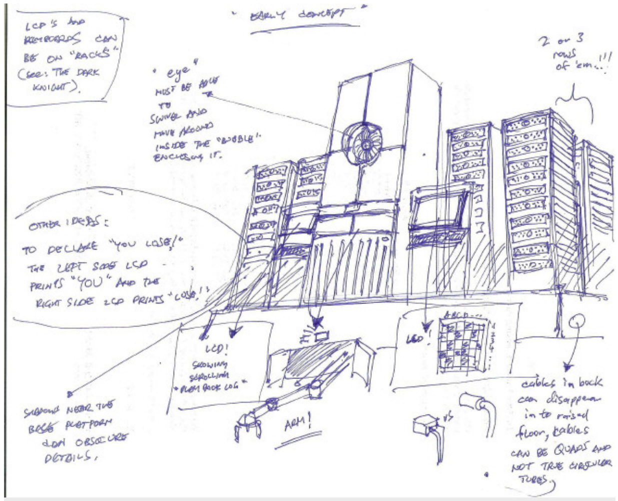
Having said that, the Mesh Deform system is noted to have a few quirks, apparently it must be the top-most Modifier to work error-free, and in addition there appears to be a minimum clearance distance required between the Mesh Cage and the Actor Mesh contained within it. Actor vertices too close to the cage will simply not respond to Mesh Cage deform. The above conditions apply particularly when a Subsurf Modifier is also used. Subsurf must be bottom-most modifier as a rule-of-thumb.

A Successful Acting Reel and Screen Tests will most likely determine the end of Pre-Production for this element in "Game Theory" with the exception of finding time and resource to attempt using layered maps for veins, displacement, normal and bump.

RSM-2009 MONOLITH Super-Computer

The story in "Game Theory" would not be complete without its antagonist. Ironically, the story technically has none since it can be argued that a Super-Computer that plays Chess is really just a prop. The true position of the Monolith Super-Computer in the story of "Game Theory" was the subject of many discussions between William and I and we kind of agreed to direct the design in a state of limbo. The Monolith might hint it has "character", but ultimately will resemble a giant appliance.

If the Human Actor ascribes to the rule of Feedback, the Monolith, while also being subjected to test feedback, is a more concrete example of The Beauty of Compromise. In my original vision, the Monolith was just a single cabinet resembling a coin-operated Arcade machine. It is through the synergy with William that ultimately I was persuaded that the Monolith would be a more meaningful presence in "Game Theory" if it was larger and more immense; a representation of the great odds talented people like to place in front of them. The change called for new Concept Art which I drew in the space of a short time to present to William for Modeling.


"EARLY CONCEPT"
LCD'S AND KEYBOARDS CAN BE ON "RACKS" (SEE: THE DARK KNIGHT).
"eye" MUST BE ABLE TO SWIVEL AND MOVE AROUND INSIDE THE "BUBBLE" ENCLOSING IT.
2 or 3 rows of 'em..!!!
OTHER IDEAS: TO DECLARE "YOU LOSE!" THE LEFT SIDE LCD PRINTS "YOU" AND THE RIGHT SIDE LCD PRINTS "LOSE!"
LCD! SHOWING SCROLLING "PLAY BOOK LOG"
LCD!
ABCD...
ARM!
VS
SHADOWS NEAR THE BASE PLATFORM CAN OBSCURE DETAILS.
cables in back can disappear in to raised floor, cables CAN BE QUADS AND NOT TRUE CIRCULAR TUBES.
Figure 4: Early Rough Concept .

If any of you love that Eye in the central unit, that one is purely William's idea.

A similar process was taken up for the Monolith's arm hatch, which no longer exists in its original state; as well as the arm it will use to play Chess with. While no real-life Chess computer was ever equipped with the robotics to physically move the pieces itself,

by Giancarlo Ng

we wanted to work in a fantasy element to "Game Theory" to enhance the "mano-y-mano" nature of the Story.

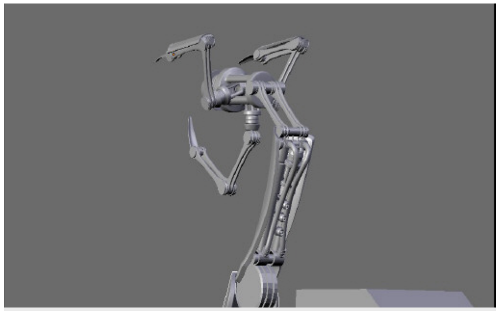


Figure 5: Rear View of Early Arm Build.

Even as a fantasy element, the Monolith still went through a rigorous research phase where we studied and compared the aesthetic shapes and colors of various real-life computers, factory arm robots, including actual machines IBM built to play Chess against Kasparov. Concept Art was developed from this pool of research photos and William went straight to building it in Blender.

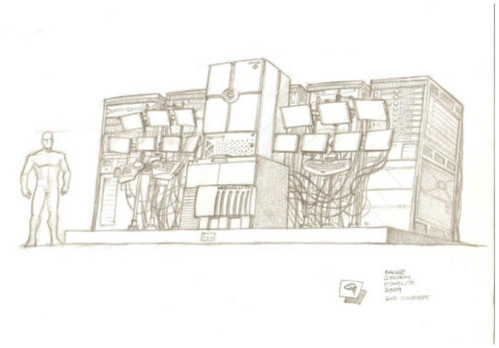


Figure 6: Refined Concept Art Submitted for 3D Mod-

There were a few re-writes going on in Script and Storyboard as a result of the change in size of the Monolith, but as we felt the Monolith's position in the Story was secure ,we gave ourselves the go-ahead to build it alongside development of the Human Actor.

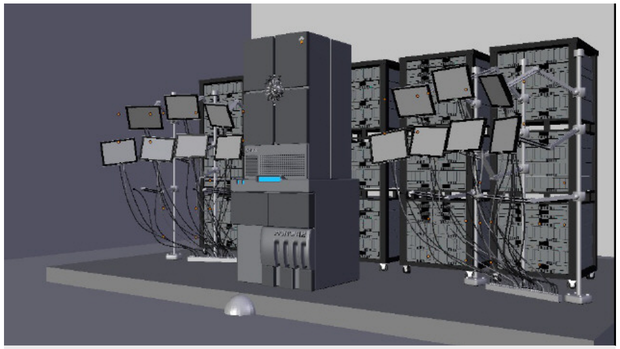


Figure 6: Monolith Super-Computer in Blender Workspace Environment.

Other than that, the Monolith uses straight forward solutions for all its moving parts and hydraulics. It is mostly composed of simple rigging for the arm and tracking constraint solutions for other moving parts.

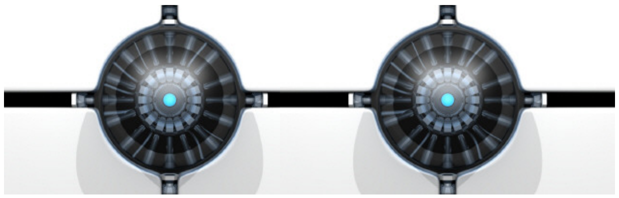


Figure 7: Tubes with Tracking Solutions for the Monolith's Eye.

by Giancarlo Ng

The Set

"Game Theory" takes place in a single non-descript room in a non-descript location. Similar to the Monolith and the level of quasi-realism placed on the Human Actor, we agreed that the Set could take that one step further by looking somewhat real and yet somewhat unreal. Last year, I conceived the room as a pure white space similar to Apple's TV ads. In January we tried early tests with a white "brick-and-mortar style" room similar to Sony's early Playstation 3 ads.

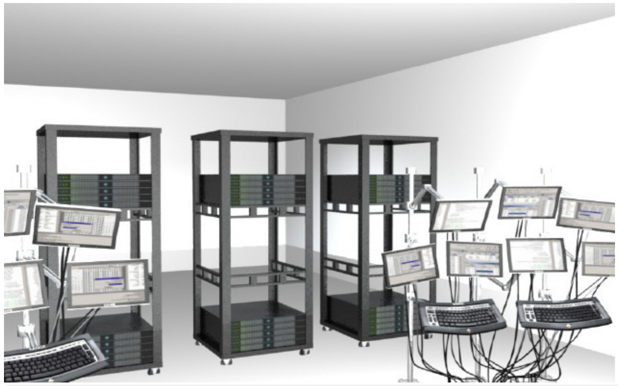


Figure 8: Rear View of Early Arm Build.

Screen tests however with the Monolith unit, showed that the color contrasts were too strong and we decided the Story was best assisted with a "neutral" room which was based on bunkers and other underground rooms such as the kind seen in films like "Panic Room", "The Dark Knight", and "The Day The Earth Stood Still".

Screen-Testing & Test Renders

In the film "Wall Street", takeover artist Gordon Gecko says to his young protégé that the secret of success is to

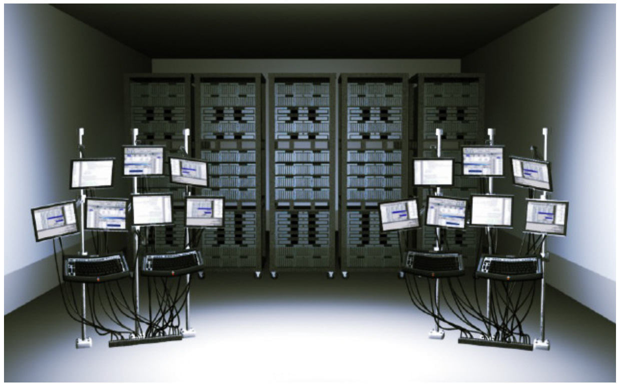


Figure 9: Early Neutral Room Test with Temp Lighting.

always know the outcome beforehand. "Every battle is won before it is ever fought" says Gordon. That is the same wisdom that applies to Screen-Testing and Feedback Testing.

The challenge as with any animation project is that ultimately one needs to project an image in only two dimensions combining colors, shapes, and sounds to try and transmit "something" to an audience. It is, frankly, not enough that one has modeled, textured, UV-ed, Mesh Cage-d, or completed some other process.

The question is always: "Does it 'work' on-screen? Do I like what I'm seeing?"

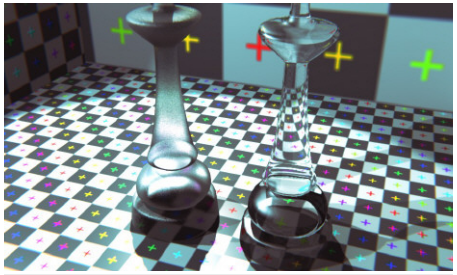


Figure 10: Early Test for Glass Chess Pieces.

This element is that part which Blender tackles by use of its Internal Render Engine and Compositor Nodes and it is in Test Renders and Screen Testing where a Team can play with these settings until a Director's vision is fulfilled.

Every Material Setting, every Mesh, every animation solution, must be tested in an environment using the lighting conditions and compositor settings which are being prepared as elements approximating the "final look" of a picture.

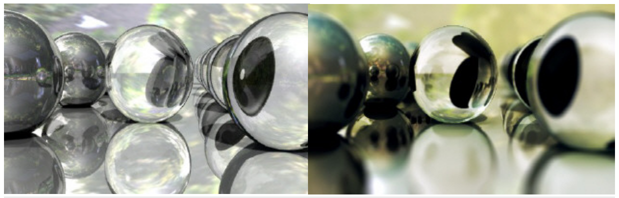


Figure 11: Illumination and Transparency Rendering Tests .

Screen-Testing is also an exercise in objectivity and courage. It sometimes takes a personal leap of bravery to admit that something you have spent weeks to create "does not work" and has to be sent back into assembly. But it is sometimes in moments of rejection like these where a project is saved from an at times merciless audience.

It is for this reason that our Fall 2009 date remains a "hopeful", but this is also why William and I are very guarded in our approach to taking risks, limiting the play length to under 3 minutes and why I chose this particular story to move forward in learning about Blender and collaborative projects.

PRODUCTION

Assuming all modeling, texturing, and testing completes successfully, and so far we have been very fortunate in this regard, "Game Theory" is expected to go into Animation of Final Scenes within the Next Quarter, a bit ahead of schedule, but that at least will give us enough time to deal with nasty surprises should they emerge.

Closing Remarks

If you like what you've read here and would like to collaborate with us in the future, there's always the next project for us to work on together. And in keeping with our vision here at our little Creative Guild, each project prepares us for the next more ambitious project 🟥

Giancarlo Ng

e-mail: cgipadawan@gmail.com

William Oglesby

e-mail: eggyolkstudios@gmail.com

MAKING OF: Brazilian Open-series

by Quadro-Chave

Vida Cotidiana - Episódio 1: A Gota (Daily Life - Episode 1: The Drop) is a free, short animated film that combines techniques of hand-painted watercolor and 3D computer graphics.

The film is being produced by Quadro-Chave, is being developed with free programs, and is licensed under Creative Commons.

The production team is working with free softwares, such as Blender, Yafaray, Gimp, Pencil and Arbaro.

We are working with Global Illumination using the Path Tracing method for final render. The resulting visuals with the textures occurred as planned, a good blend of organic painting and 3D computer graphics environment.

```
YafaRay (289)    render time: 9m 21.52s    intel-core2-3,5ram
AA passes: 1, AA samples: 5/1 (box)  Raydepth: 10 Shadowdepth: 10
Lighting:  GI: photons (1600000), bounces: 20 with background
```

For the first person cameras, we are using Voodoo Camera Tracker software to generate a natural camera movement. The movement of data from the camera will be imported into blender. We used a low-cost camera to film, because we only needed to capture the movement, not the image.

This type of technology is also a solution for smaller studios or anyone who has willingness to work with them.

This episode is the beginning of the animation mini-series, Daily Life.

The Quadro-Chave's intention is to bring the elements and events of our daily life and create small animated polls. Other episodes are in the process of creation and organization of ideas. As a independent work and open movie, in this case an open series, our team is always looking for help, any kind of support would be good. If you are interested in participating in episodes, or simply from some event in your daily life that you feel is worth being portrayed, just contact us through the site in the contact section. We look forward to receiving your contribution.

All the research results that were developed during the production of the episodes, as well as tutorials teaching how we did the animation step by step, will be available on dvd and launch the official website of the animation ■

www.vidacotidiana.quadrochave.com

By Paul Fitzpatrick

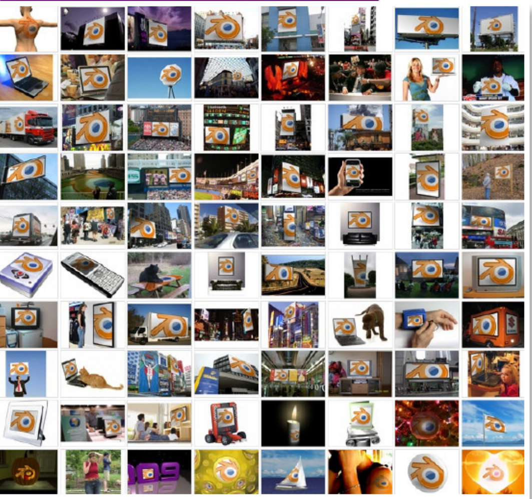


Figure 1

Introduction

I enjoy using Blender to make fun little pictures and animations for birthdays or holidays. I imagine I'm not the only one. One day I thought: wouldn't it be neat if you could put Blender designs online in an easy-to-use form, so that anyone (not just Blender users) could customize them for their own use? This idea slowly became a website called MakeSweet, and I'd like to talk about how it evolved and what I learned along the way.

Figure 1: A selection of designs from MakeSweet. The designs take an image supplied by the user (here I used the Blender logo) and produce a picture or animation based on it. There are some soft body animations, lens effects, and lots of photo backdrops with 3D surfaces overlaid. 90% of the designs shown here are user-created (the rest I made).

The world of online generators

Online tools that help you make things are called generators. For example, there are generators that help you make buttons, write love letters, construct plausible excuses, and so on. Within the world of generators, visual effects are popular. Who wouldn't want to put their photo on currency, or see it on a billboard?

When I looked around, I found there were already plenty of sites out there for creating visual effects. But I guessed that the openness of Blender might let me go further than other similar sites. Adapting a free and open source tool to new uses is much more pleasant than working with closed products, because there are fewer roadblocks to integration. It is also a wonderful base to build a community around, since it can be freely shared. So I hoped that, if I did it right, I could enable a community of generator-makers, rather than just create generators myself. And in fact that has started to come together (see examples in Figure 1), though in a form I didn't anticipate. I'll explain, but first let me talk about the basic technical and artistic challenges of using Blender to make generators.

The technical challenge

Blender is a wonderfully adaptable program used by everyone from movie-makers to architects. To adapt it for online use, the main problem is time. It takes time to render pictures and animations, but on the internet there is a strong expectation of immediate response. Let's think about what properties a generator site needs to have:

1. Fast to respond. Something cool must happen within seconds of a user's request.
2. Cheap to run. The site must not cost too much to operate.
3. Scalable. The site must remain fast to respond and cheap to run as its popularity grows. Otherwise the only possible fates for the site would be to either self-destruct or remain forever obscure, neither of which is particularly desirable.

To satisfy these requirements with Blender, we're going to need to be ruthless about baking, caching, and otherwise pre-computing everything we possibly can. Ideally, we would also offload as much computation onto users' own computers as possible

by Paul Fitzpatrick

There is a Blender browser plug-in that can be made to work, but expecting users to install software in order to use a generator site is a bit too much to ask. The most practical solution is to do all the Blender work on a server (or set of servers) and send the results as images or animations to a user's browser. So we really need to reduce the amount of work done for each user request to the bare minimum.

How far we can go with pre-computing depends on what kinds of user customizations we want to support. I decided to restrict my attention to a class of designs and customizations that was wide enough to allow interesting generators, but restricted enough to allow almost everything to be pre-computed. The customizations I decided to support were the replacement of image textures with a new image of the user's choosing. This is a good match for user expectations ("upload a photo and see something fun happen"). The designs I decided to support were those in which the image textures under user control do not affect geometry and do not overlap. This rules out some possibilities, for example 3D text or displacement maps, but still leaves a range of useful effects, as we'll see.

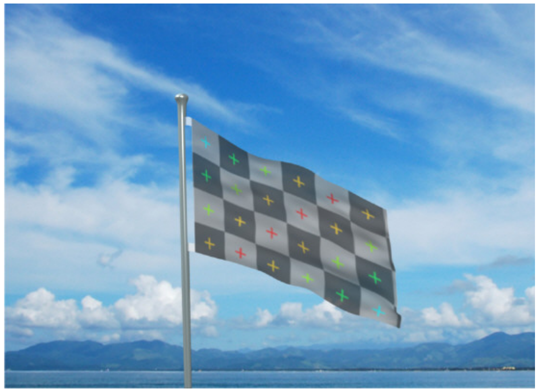


Figure 2: *An animated flag design with one image texture under user control (the surface of the flag). Prior to online use, the coordinates that a test image projects to in a rendered result are probed and recorded, for each frame in the animation. This allows fast stills and animations to be generated when a user supplies a photo.*

To get technical, the requirement I chose was that the appearance at each location in a frame of the output render should be a function of the appearance of at most one location in the input image (I'll also assume the function is linear or can be approximated as linear, although this isn't as important). Consider the flag design in Figure 2. Here, the sky is fixed – it is not affected by the input image.

The appearance at each location of the render within the flag's boundaries is drawn from a single location in the input image, although exactly which changes from frame to frame as the flag billows. Under these conditions, it is easy to get Blender to pre-compute how the output render is affected by the input image, and to store this mapping efficiently.

This can be done at leisure, offline. Online, then, for each user request, I use a stripped-down optimized renderer I wrote called the "Mixer" (meant to sound like a cheap generic knock-off of "Blender") to apply this mapping to the user-supplied image without needing to run Blender. The quality is not quite as good as Blender would give due to sampling issues, but with it, users then have a good sense of whether the generator is giving them something they want, and can go on to make a high-resolution version or an animation. For a high-resolution result, I expect that users will be sufficiently motivated to be willing to wait for a few more seconds, so I run Blender on the server for them (at a low priority so as not to hurt response time for other users).

For animations, I generate an animated GIF produced by concatenating preview results; using Blender for an animation would take a very long time, and the extra quality of renders would not be worth much once jammed into a 256-color GIF palette (also, individual frame quality matters less in a moving sequence).

There are lots of other possible choices one could make to get an efficient family of generators. This is just the one I chose to start with, and it has worked out pretty well in practice. The key point is to support a wide enough range of effects to be interesting, but be constrained enough to allow fast rendering (see Figure 3).

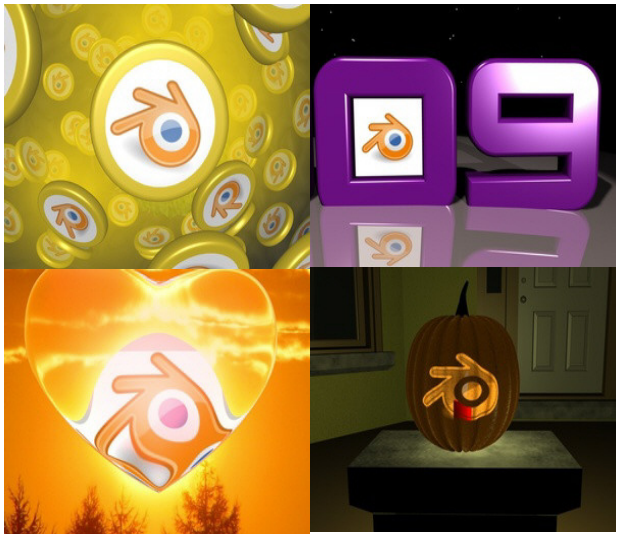


Figure 3: For speed, MakeSweet restricts designs to cases where each location in the output is a function of at most one location in an input image. There's plenty of scope for creativity within the limits of what the website can quickly render. It is fine for the user's image to appear multiple times, either by having the same material on many objects (see top left, from a St. Patrick's day pot of gold animation) or by reflection (see top right, from a New Year's day animation). Distortions are also fine (see bottom left, from a Valentine's animation, where the user image is seen through a heart-shaped lens). And transparency can work too (see bottom right, a pumpkin carving from a Halloween animation).

The artistic challenge

So far I've just talked about the technical side of making generators. But all that is worth nothing without good art that people enjoy. The principle artistic lesson I have learned from working on MakeSweet is that I am not an artist. There is an interesting challenge in creating good generators with a “narrative” that users identify with. Here are some of the difficulties of the medium, at least as implemented on MakeSweet:

- The generator must have some flexibility to accommodate unknown user input. You don't control the whole story.
- For animations: you have just a few seconds to tell the story. The longer the animations, the more CPU time burned producing them, and the fewer people will stick around to create them.
- Resolution, frame-rate, and palette are limited (especially if GIF animations are used).

Here are some styles of generator that work:

- Simple scenes based on holidays such as Halloween, Valentine's Day, and so on. These occasions have a lot of shared cultural knowledge to build on.

by Paul Fitzpatrick

by Paul Fitzpatrick

- Scenes where people already expect to see messages — billboards, monitors, televisions, tattoos, flags, signs, etc.

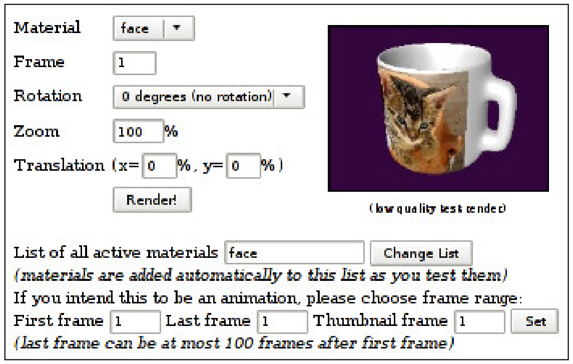
Sometimes generators fail to attract interest if they are not sufficiently centered on the user input, or don't really have some root in popular culture that helps people grasp them instantly. For example, the heart design in Figure 3 is part of an elaborate animation of falling rain drops that turn out to be heart shaped. It ended with a zoom-in on a drop magnifying the sun, which had the user's image overlaid on it. It was intended for Valentine's day, and wasn't a complete flop, but neither was it a great success. People instead sought out a much simpler design from the previous year, an animation of a locket in the shape of a heart opening to show two photographs. That animation had a much clearer narrative and hook for the user message.

The community challenge

My hope with MakeSweet is that, by solving the technical problem of making generators, I could support people with actual artistic talent in their creation of generators online, for fun or as part of an interactive portfolio. I believe Blender is a great choice for this. To see why, let's look at the procedure I ended up using to make generators with Blender online:

1. First, I design an interesting .blend that does something neat with an image texture of my son (my standard test image).
2. I make a short configuration file that specifies the image texture (or textures) that should be customizable by the user.
3. I have Blender pre-compute a mapping from an input image (or images) to a render.
4. I upload the mapping to the website. Done!

The first two steps require no special skills beyond a knowledge of Blender. And the remaining steps can be made fully automatic. So there's no reason why anyone who knows Blender shouldn't be able to make generators. And in fact, I created a service on MakeSweet called the Generator Wizard for doing just this. It is in testing right now, and I encourage anyone interested to give it a try (http://makesweet.com/wizard).

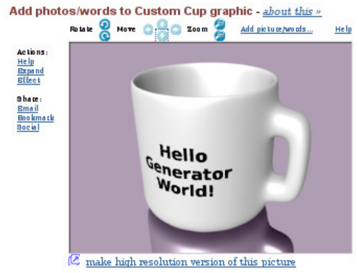


Material: face
Frame: 1
Rotation: 0 degrees (no rotation)
Zoom: 100 %
Translation (x= 0 %, y= 0 %)
Render!

(low quality test render)

List of all active materials: face — Change List
(materials are added automatically to this list as you test them)
If you intend this to be an animation, please choose frame range:
First frame 1 Last frame 1 Thumbnail frame 1 Set
(last frame can be at most 100 frames after first frame)

Figure 4: *The Generator Wizard lets you upload a .blend file and convert it into an online generator. For example, here we upload a cup model with an image texture on its surface (where the cat is), and what we get is a webpage that lets anyone replace that texture with their own picture or words. Code is also provided for embedding the generator on other websites.*



Add photos/words to Custom Cup graphic - about this »

by Paul Fitzpatrick

With the wizard, we have entered the world of "generators of generators" – we've made a tool that converts user supplied material (a .blend) into a generator. We can push this idea further, and develop other generators of generators (let's call them "GOGs") with the following trick: the user provides parts that the site assembles into a .blend, and then a generator is made from that .blend as before. So far I've had most success with a "Billboard Generator of Generators." This is a small Flash widget that lets a user select a flat surface

The widget does this by computing the 3D location of the surface and then generating a simple .blend file that does the necessary projection.

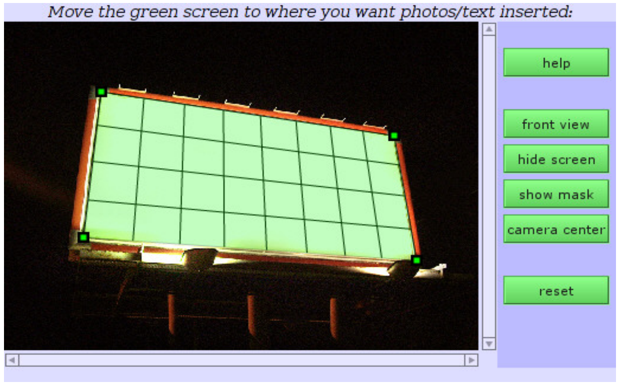

Move the green screen to where you want photos/text inserted:
help
front view
hide screen
show mask
camera center
reset

Figure 5: The Billboard GOG. Users upload a photo of a billboard or any other object with a flat rectangular surface, and then identify the corners. A .blend is then assembled with their photo in the background and the appropriate 3D plane overlaid on it. A generator can then be made.

Users can advance by learning to apply masks for flat surfaces that are not rectangular (if you look closely, there are several examples in Figure 1). For anything more complicated, they are nudged towards learning Blender. The billboard generator has proven popular, and scenes made with it by users now dominate the site.

Conclusion

It turns out that Blender is a great file format for expressing visual generators. It nicely separates out the artistic work from the geeky integration. And on that geeky side, Blender is a joy to integrate, and plays very well with other software. MakeSweet was a lot of fun to put together, and has already been a lot more successful than I dared wish for. I hope it will help expose Blender to some of the vast horde of people out there who would love to play with 3D, but haven't yet realized that there's really nothing stopping them anymore. Two years ago, I was one of them.

Paul Fitzpatrick

My day job is the RobotCub humanoid robotics project, based in Genoa Italy. Our humanoids have completely open source hardware designs and software.

Website: paul.giszpatrick.com

MAKING OF: Anti Chamber (Antechamber)

Anti Chamber a 3d game level

By Ken Beyer

Blender was used as a production tool to produce the modelled assets for this entry into the inaugural Unearthly Challenge competition between Game Artisans and Poly-count. The theme for this first challenge was “Good v Evil”; I chose “evil” but went for a "spooky" evil which allowed the use of a blue / green palette of colours rather than the expected red / black route.

Once modelled and composed in Blender 3D, the 3D assets were then exported as *.ase models and imported in Quake 4's content editor, Radiant, to be built up as a level; lighting, materials and effects are placed, the level compiled and then loaded into Quake 4 itself as a custom single player map. Textures were photo-sourced and normal maps generated using Crazy-Bump.

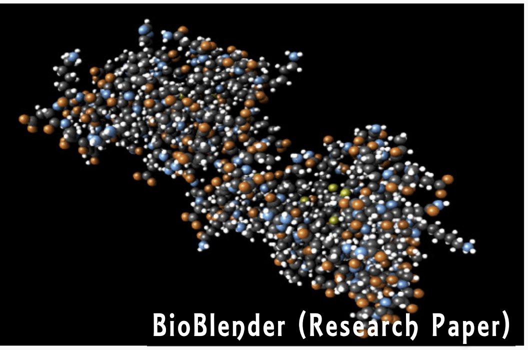
Tools used

- Blender 3D (v2.45)
- Corel Photopaint
- CrazyBump
- Quake 4
- Digital camera

Read the full post-mortem here:
http://www.katsbits.com/cgi-bin/ikonboard.cgi?act=ST;f=3;t=80 ■

Ken Beyer ('kat')

Web site – http://www.katsbits.com
e-mail: info@katsbits.com



by SciVis IFC - CNR

Introduction

If we all are alive, it's because we have unnumbered nano-machines that work inside our cells; these machines, through an unbelievable network of reactions, make us breathe, walk, think, eat and all other things told and untold.

At the nanoscopic level (where the unit of measurement is one billionth of a meter, 10-^{9}), most life forms, including plants and bacteria, are similar and beautiful. That is, if you could see them. Scientists devolve their work to understand how all this happens, and have gathered enough information to start putting together a complex picture. Actually, a motion picture would be better...

Blender in Biology?

When we think of a biological lab, most of us imagine scientists in a lab-coat, pipetting and mixing a number of mysterious substances, possibly with some smoke around. While this is not unreal (except the smoke), modern biology also heavily counts on the elaboration of data extracted in the classical laboratory. This is how scientists have built databases of 3D coordinates of many (>50 thousand) of the small objects at work in the cells, a database which is available for use by everyone.

We have decided to use Blender for manipulating these data, with the aim of showing the molecules, their activities, their physical and chemical properties and the environment in which they operate. Because not everything is known of the list above, we also use the Blender Game Engine as a tool for research.

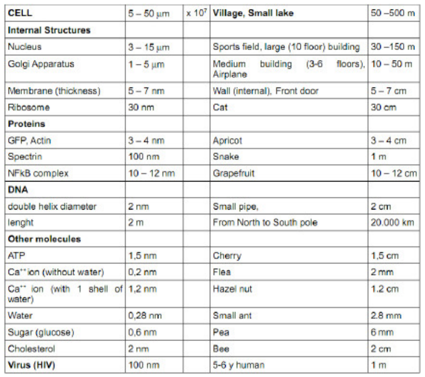
At the moment BioBlender is a collection of scripts, tools and procedures built on Blender 2.48a. We also use the experimental version SuperTexture Node and the recorder developed by Ash (Osobná Stránka).

The project is by no means concluded, and we will be happy to consider ideas, improvements and suggestions from the very collaborative Blender community, to which we are grateful for the great help we have already received.

This article is an introduction to our work, and will guide you through the microscopic world of cells explaining some of the steps we have made towards the construction of BioBlender.

Cellular landscape

Cells, the fundamental unit of life, contain a full microscopic world. A good idea of the relations that take place within cells can be obtained by 'exploding' a typical cell about 10 million times, to a size with which we are familiar (see Table).



CELL	5 – 50 µm	x 10^7	**Village, Small lake**	50 –500 m
Internal Structures				
Nucleus	3 – 15 µm		Sports field, large (10 floor) building	30 –150 m
Golgi Apparatus	1 – 5 µm		Medium building (3-6 floors), Airplane	10 – 50 m
Membrane (thickness)	5 – 7 nm		Wall (internal), Front door	5 – 7 cm
Ribosome	30 nm		Cat	30 cm
Proteins				
GFP, Actin	3 – 4 nm		Apricot	3 – 4 cm
Spectrin	100 nm		Snake	1 m
NFkB complex	10 – 12 nm		Grapefruit	10 – 12 cm
DNA				
double helix diameter	2 nm		Small pipe,	2 cm
lenght	2 m		From North to South pole	20.000 km
Other molecules				
ATP	1,5 nm		Cherry	1,5 cm
Ca^{++} ion (without water)	0,2 nm		Flea	2 mm
Ca^{++} ion (with 1 shell of water)	1,2 nm		Hazel nut	1.2 cm
Water	0,28 nm		Small ant	2.8 mm
Sugar (glucose)	0,6 nm		Pea	6 mm
Cholesterol	2 nm		Bee	2 cm
Virus (HIV)	100 nm		5-6 y human	1 m

by SciVis IFC - CNR

If we look at a medium size cell of about 10µm size, and we compare it to a village or to a lake, not very big but very deep, all internal components can be re-sized accordingly, and we see that the nucleus, where all DNA is stored, can occupy up to half the volume of a cell and is the major internal object. Objects of this size (including the Endoplasmic reticulum, the Golgi apparatus, mitochondria, chloroplasts and some other structures) can be seen with microscopic techniques that allow us to visualize their shape and (sometimes) their dynamic activity.

It is important to note that, in contrast with the human-size world with which we are familiar, the entire volume is occupied, such that it might be easier to imagine a water body rather than one filled with air. Furthermore, we have to notice that gravity is irrelevant at this size (the mass of objects is too small to be significantly affected by the Earth gravity field), and that movements of cellular components is mostly driven by thermal agitation.

The boundary of the cell, as well as the walls delimiting internal volumes, is made of membrane, a soft, flexible and (relatively) thin double-layer that mediates transport of material and information between in and out. This is an extremely important structure that deserves more detailed description, and which we have modelled with a complex system of particles, dynamic fields, and animated procedural textures.

Going deeper and smaller, we meet nucleic acids: DNA and RNA. Everyone is familiar with the double helix of DNA, but few people realize that in relative size, if the diameter of the helix is 2 cm (a very thick rope or high tension cable), its length is 20.000 km, about half the Earth's circumference. DNA is packed in a very efficient organization that allows access to it both for retrieving information and for replicating it every time a cell divides. This organization is accomplished thanks to the involvement of proteins, the major players of cellular life, and the most immediate subjects of our animation efforts.

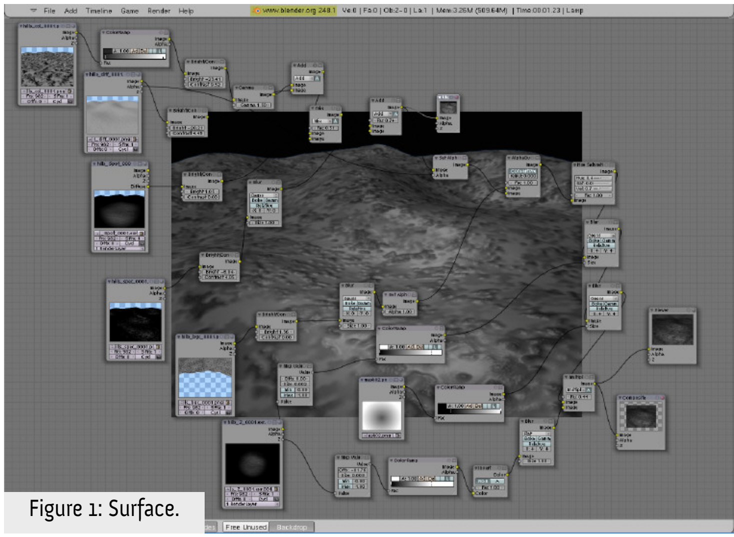


Figure 1: Surface.

From this overview, it should be clear that we can observe cellular life at many different levels of focus, spanning 4 or 5 orders of magnitude. However, if we can easily recognize the size of familiar sights (a valley or mountain, a building, a tree or an insect), there are no immediate references for attributing dimensions to objects that we have never seen before, such as ribosomes and actin

One of the tasks we face, is to provide the observers with clues indicating the scale of the objects in the scene.

by SciVis IFC - CNR

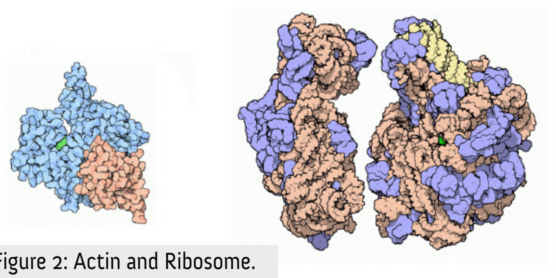


Figure 2: Actin and Ribosome.

Proteins

Because proteins are the major characters of cellular life, and indeed are a major subject of scientific studies, we developed first a system to import them into Blender. It is necessary to describe some details of their general structure to understand how they are built (in nature and in Blender) and how they can move.

Proteins are constructed as a linear sequence of amino acids, which are small assemblies of atoms that share some features that allow them to be linked directionally one after the other. There are 20 different types of amino acids, distinguished by their lateral parts (Side Chain), each composed of a specific number and connection of atoms. The linkable parts, equal for all amino acids, form the Main Chain. Each protein contains from a few hundred to a few thousand amino acids, and despite being a linear sequence, each one, immediately after being built, folds in space to acquire a 3D structure which is remarkably stable, although flexible.

The structure of proteins can be determined experimentally, and is stored in the Protein Data Bank (www.pdb.org) as a .pdb file, which contains information about the sequence of the molecules, the details of experimental procedures to obtain the structure, and the list of all atoms of the protein with their XYZ coordinates. Using this information and including the chemistry of amino acids (how atoms are connected), it is possible to build in the 3D environment the complete structure of any protein.

While X-ray crystallography results in determination of the position of all atoms with good resolution, for a single conformation, other types of techniques such as Nuclear Magnetic Resonance can yield a collection of coordinates, corresponding to a number of positions that the protein can assume. To obtain motion, all we have to do is find the path that every atom follows to go from one conformation to another, taking into account also the limitations and constraints imposed by chemistry and physics.

We describe next our work to produce such molecular motion.

PDB importer and animator

Starting from our previous work in Maya, we wrote a program to read .pdb files and build the molecule in Blender. The .pdb file of interest is fetched and read line by line. Atoms are identified for their nature (Carbon, Oxygen, Nitrogen etc.), their position (X,Y Z) and the amino acid to which they belong. These information are elaborated using a library that stores atomic connections for all amino acids.

Through the interface, shown in Fig. 4. the user can select the .pdb file, the atoms to be imported (main chain only, main and side chains, or all atoms including hydrogens), the kind of object to be built (empties, spheres, metaballs), how many conformations and in which order to import them (the .pdb file has no specific order) and the transition time between different conformations. Note that in the .pdb file every conformation is called MODEL.

Atoms are instanced to spheres, the chemical bonds are built as rigid body joints, (or bones for IK animation)

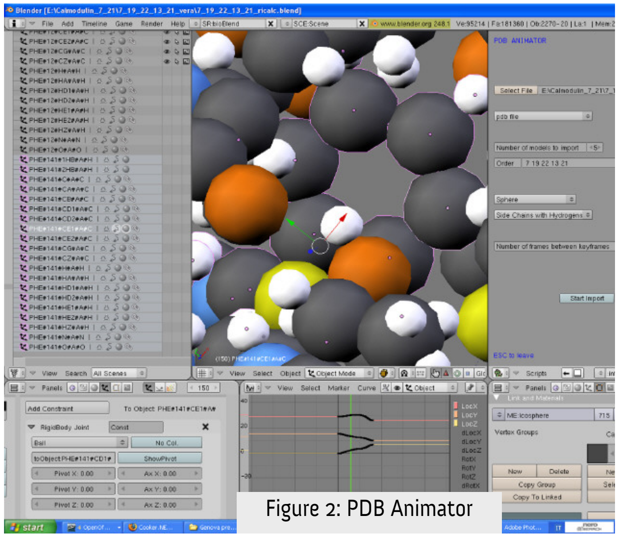


Figure 2: PDB Animator

and a keyframe is assigned to every conformation in the list. The spheres corresponding to different atoms are sized according to the atomic Van der Waals radius and have a material for visualization and a spherical collision radius (bounding box) for evaluation of motion in the Game Engine.

Once all models of interest are imported, Blender will have an IPO curve for every atom (as a consequence of having keyframes), that interpolates directly between positions at subsequent conformations. However, these will not consider the joints (that maintain fixed distance between connected atoms) nor collisions. To obtain a trajectory that includes both these features, it is necessary to play the scene with the Game Engine. The scene also contains a recorder that registers the position of atoms during the game and inserts a key frame to the atomic IPOs for every frame.

At this point the motion is set for re-playing without further calculations; we can retrieve the position of all atoms at intermediate frames (as new .pdb files) and use them to evaluate the quality of the structure in physical and chemical terms, using specialized programs, such as VMD or SwissPDB viewer.

Calmodulin

Calmodulin (CaM, Fig. 5) is a small protein (148 amino acids, about 2.300 atoms, including hydrogens) that transmits the signals arriving to the cell in the form of free Calcium ions, and delivers information to other proteins, thus activating processes such as smooth muscle contraction (useful for breathing, food processing and blood circulation) or ring contraction at the cell division. The protein is arranged spatially in two domains connected by a flexible linker.

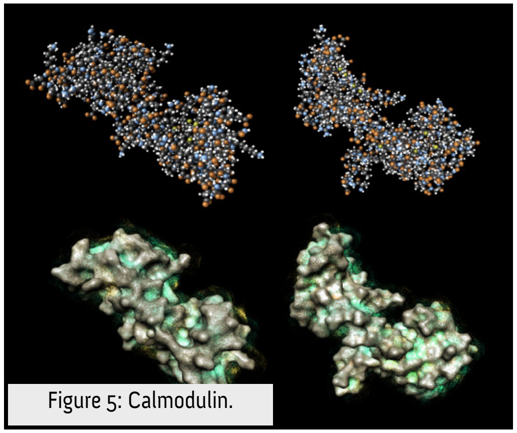


Figure 5: Calmodulin.

In the absence of Calcium, CaM is believed to stand around idling by itself. When a signal arrives, 4 calcium ions bind to specific spots in the two heads of Calmodulin and determine a major conformational change, exposing some amino acids with more lipophilic properties, which simply means that in the new form, CaM will attach itself to other molecules, thus transmitting the signal to these so-called effector proteins.

by SciVis IFC - CNR

Many studies have revealed the conformation of CaM in the empty and Calcium-bound form. We have used this protein as a the first model for the PDB Animator.

Rendering chemistry and physics

The actual aspect of objects beyond the resolution limits of our sight, is something that does not exist. Nevertheless, it is possible to represent the space occupied by the atoms of the molecule, and to attribute to its surface visual properties to indicate some of its behavioral features. At nanometer scale, concepts such as color, brilliance, roughness, opacity and so on have no meaning; instead we face properties such as pH (acidity, or proton concentration), electric potential, hydropathy, oxidizing or reducing power and others.

Among the most relevant properties that affect molecular behavior are the Electric Potential (EP) and the Molecular Lipophilic Potential (MLP) that indicate the tendency of a surface to attract or repel other charged molecules, and the affinity or repulsiveness for water, respectively. In an effort to display the behaviors associated with EP and MLP of the molecules, we have performed some steps that permit to import values in Blender, as schematized in Fig. 6.

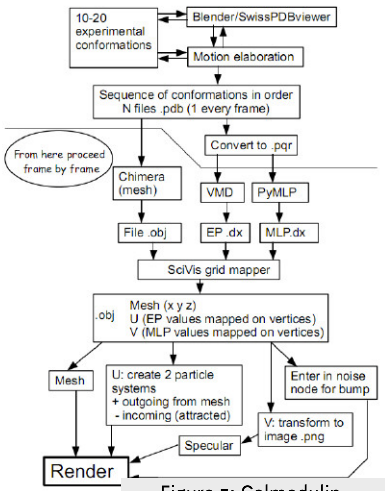

10-20 experimental conformations
Blender/SwissPDBviewer
Motion elaboration
Sequence of conformations in order N files .pdb (1 every frame)
Convert to .pqr
From here proceed frame by frame
Chimera (mesh)
VMD
PyMLP
File .obj
EP .dx
MLP.dx
SciVis grid mapper
.obj
Mesh (x y z)
U (EP values mapped on vertices)
V (MLP values mapped on vertices)
Mesh
U: create 2 particle systems
+ outgoing from mesh
- incoming (attracted)
Enter in noise node for bump
V: transform to image .png
Specular
Render

Figure 5: Calmodulin.

The visualization of the forces generated by electric charges, and exerted to the surrounding medium (water, which is dipolar itself, some ions and eventually other proteins), has been solved using a particle system, with sparks going from the surface out for positive values, and being attracted to the surface for negatives. The MLP is seen as a property of the surface: smooth reflective material for lipophilic and rough, more dull for hydrophilic.

Work in progress

Our work has more than one scope:

- protein motion is still a major subject of studies in molecular biology: if Blender can be used to provide an approximate solution, while avoiding extreme calculations, this might develop as a research instrument with important uses by biologists, chemists and other scientists. Also other cellular components (DNA, RNA, sugars chains, small molecules, etc.) can be modelled and animated using similar principles;
- the power of images for explaining (and understanding) can be exploited in schools of all levels, from primary to postgraduate, and can also be useful for exploring new hypotheses during theoretical elaboration;
- the possibility of observing complex scenes with many components at different scales, can enable a deeper understanding of cell behavior;
- the availability of different images from inner life can be inspirational to artists, whose work and insights are important for artists themselves, scientists and also everyone else;
- finally, not least, the developments we are adding to Blender might be interesting also for other Blender users, who can use them for whichever creation they can think about.

Altogether, we encourage all scientists and artists to use the instruments we have developed, to produce new images and animations, and to refer to us any problems or suggestions. For this purpose we have opened a blog in the website of one of the major scientific journals, Nature, which can be reached at this address. We hope that this blog will become a place where biologists, artists and graphic scientists meet and discuss, and continue building BioBlender as a resource for everyone.

Needless to say, as soon as our scripts will be presentable and reasonably stable, we will deposit them for download (now we distribute them on request), and will record a tutorial to explain the use.

We are grateful to the Regione Toscana, Italy, for a major funding that made this work possible.

The Scientific Visualization Unit (www.scivis.ifc.cnr.it)

- Raluca Andrei (Molecular Biology PhD student, Scuola Normale Superiore)
- Marco Callieri (Informatics researcher at ISTI - CNR)
- Ilaria Carlone (Biologist at IFC - CNR)
- Claudia Caudai (Mathematician IFC - CNR)
- Stefano Cianchetta (3D Graphic at IFC - CNR)
- Tiziana Loni (3D Graphic artist at BBS)
- Yuri Porozov (BioInformatics researcher at IFC - CNR)
- Maria Francesca Zini (Chemist and Programmer at IFC - CNR)
- Monica Zoppè (Biologist at IFC- CNR)

Figure 1. The surface of the cell is seen as a series of mobile hills, covered with different groups of proteins, saccharide chains and lipids. The primary pattern was developed (in Maya) with a system of 5 different kinds of particles (10.000 in total), to which various per particle dynamic features were attributed. The system, which also included some boundary conditions and random turbulence origins, was played for 500 frames, recorded and rendered giving to the particles blobby (metaball) features and colors as grey scale. This rendered animation was used as texture source in the nodes of Blender, using the Blender SuperTextureNode (http://www.graphicall.org/builds/builds/showbuild.php?action=show&id=862). The image shows a screenshot of the final compositing pass.

Figure 2. Example of some cellular components: Actin, on the left, is a medium size protein, composed of 375 amino acids, very important for cell (and organism) motility. Much of the protein component of animal muscle is actin. Ribosomes, right, are complex machines made up of over 30 proteins and 3 RNA components, and are made of two parts. They are the factory where proteins are constructed by linking amino acids in series, as instructed by the nucleic acid mRNA. The mass of the ribosome is about 1000 times the mass of one actin molecule. The Images are from the Molecule of the Month in the PDB website, by David S. Goodsell.

Figure 3. Amino acids

Figure 4. PDB Animator

A screenshot of the PDB Animator and of a detail of a protein in 'working mode', where atoms are all drawn as spheres of different colors, and the surface is not calculated.

by SciVis IFC - CNR

Figure 5. Calmodulin

The atomic structure and rendering of two different conformations of Calmodulin. On the top we see all atoms, colored by atom identity (Carbon, Nitrogen Oxygen and Hydrogen). We use this kind of visualization to work on motion and to study the molecular structural properties.

On the left CaM is Calcium-free, and on the right, Calcium-bound. Structural changes are not very large, yet the surface properties undergo a major transition; notice the shiny spots, that indicate protein activation: this is where it will make contact with other downstream effector proteins, thus effectively transducing the signal within the cell. The rendered images are shown with green and yellow colors to indicate polarity of the electric field, but during animation the colors are not necessary, because the particles travel towards or out from the protein surface.

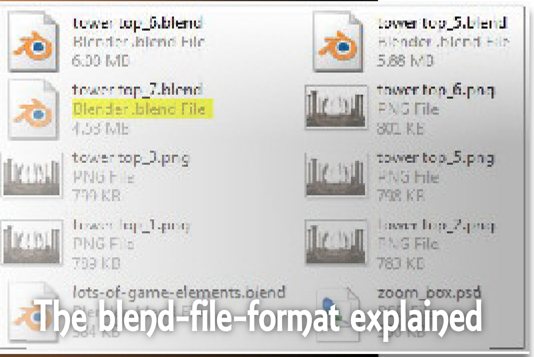
Figure 6. Protein rendering flow.

After motion is calculated with the Game Engine, each frame is stored as a .pdb file, sent for checking by chemical and physical programs, and reimported back in Blender for rendering.

The .pdb file is converted to .pqr, through a program that associates the appropriate values of partial charges to every atom, according to its properties as inferred from the .pdb information and to libraries that store values determined experimentally. This step is performed once for all conformations attributed to a molecule (i.e., the electric values associated to each atom do not change with its position). This file is sent to the molecular program VMD, and the module APBS electrostatics is executed. This module solves the Poisson-Boltzmann equation on a discrete domain represented by a grid which extends around the molecular surface. The molecule surface mesh is saved as a VRML file and the EP, calculated in each cell of the regular grid, is saved in a simple ASCII file (EP.dx). At the same time, another program is used to calculate and map the Molecular Lipophilic Potential, which stores data in MLP.dx file.

These data files are used with a home made program (SciVis grid mapper) to map values from the grid to the surface, and are then stored in a new .obj file that can be easily read by Blender: EP values assigned to vertices are stored as U values, while LMP is stored in v field.

Once the Potential data have been read inside blender and mapped on the surface, they are transformed in grey scale textures which are used for setting the grades of specularity and the frequency of bump (for MLP) and for generating the particles that indicate EP 🟥



The blend-file-format explained

by Jeroen Bakker

Introduction

I'm working on a product that integrates Blender into a render pipeline by using the Blender command line and blend-files (.blend). The command line is not a problem as it is commonly used, but using blend-files outside Blender is difficult, because it is not that well documented. On the Internet, I've only found some clues about it on the Blender architecture pages [ref: http://www.blender.org/development/architecture/] and these were not sufficient. To really understand the file format, I had to go through Blender's source code. In this article I will describe the blend-file-format with a request to tool-makers to support blend-file.

First I'll describe how Blender works with blend-files. You'll notice why the blend-file-format is not that well documented, as from Blender's perspective this is not needed. We look at the global file-structure of a blend-file (the file-header and file-blocks). After this is explained, we go deeper to the core of the blend-file, the DNA-structures. They hold the blue-prints of the blend-file and the key asset of understanding blend-files. When that's done, we can use these DNA-structures to read information from elsewhere in the blend-file.

In this article we'll be using the default blend-file from Blender 2.48, with the goal to read the output resolution from the Scene. The article is written to be programming language independent and I've setup a web-site for support.

Loading and saving in Blender

Loading and saving in Blender is very fast and Blender is known to have excellent downward and upward compatibility. Tom Roosendaal demonstrated that in December 2008 by loading a 1.0 blend-file using Blender 2.48a [ref: http://www.blendernation.com/2008/12/01/blender-dna-rna-and-backward-compatibility/].

Saving complex scenes in Blender is done within seconds. Blender achieves this by saving data in memory to disk without any transformations or translations. Blender only adds as file-block-headers to this data. A file-block-header contains clues on how to interpret the data. After the data, all internal Blender structures are stored. These structures will act as blue-prints when Blender loads the file. Blend-files can be different when stored on different hardware platforms or Blender releases. There is no effort taken to make blend-files binary the same. Blender has created the blend-files in this manner since release 1.0. Backward and upwards compatibility is not implemented when saving the file, this is done during loading.

When Blender loads a blend-file, the DNA-structures are read first. Blender creates a catalog of these DNA-structures. Blender uses this catalog together with the data in the file, the internal Blender structures of the Blender release you're using and a lot of transformation and translation logic to implement the backward and upward compatibility. In the source code of blender there is actually logic which can transform and translate every structure used by a Blender release to the one of the release you're using [ref: http://download.blender.org/source/blender-2.48a.tar.gz....]The more difference between releases the more logic is executed.

by Jeroen Bakker

The blend-file-format is not well documented, as it does not differ from internally used structures and the file can really explain itself.

Global file-structure

Let us look at the global file-structure. A blend-file always start with the file-header followed by file-blocks. The default blend file of Blender 2.48 contains more than 400 of these file-blocks. Each file-block has a file-block-header and data. This section explains how the global file-structure can be read.

File-Header

The first 12 bytes of every blend-file is the file-header. The file-header has information on Blender (version-number) and the PC the blend-file was saved on (pointer-size and endianness). This is required as all data inside the blend-file is ordered in that way, because no translation or transformation is done during saving. The next table describes the information in the file-header.

reference	structure	type	offset	size
identifier	char[7]	File identifier (always 'BLENDER')	0	7
pointer-size	char	Size of a pointer; all pointers in the file are stored in this format. '-' means 4 bytes or 32 bit and '_' means 8 bytes or 64 bits.	7	1
endianness	char	Type of byte ordering used; 'v' means little endian and 'V' means big endian.	8	1
version-number	char[3]	Version of Blender the file was created in; '248' means version 2.48	9	3

Endianness addresses the way values are ordered in a sequence of bytes [ref: http://en.wikipedia.org/wiki/Endianness]. Blender supports little-endian and big-endian. In a big endian ordering, the largest part of the value is placed on the first byte and the lowest part of the value is placed on the last byte. In a little endian ordering, largest part of the value is placed on the last byte and the smallest part of the value is placed on the first byte. Example: writing the integer 0x4A3B2C1Dh, will be ordered in Big endian as 0x4Ah, 0x3Bh, 0x2Ch, 0x1Dh and be ordered in little endian as 0x1Dh, 0x2Ch, 0x3Bh, 0x4Ah.

The endianness can be different between the blend-file and the PC you're using. When these are different, Blender changes it to the byte ordering of your PC. Nowadays, little-endian is the most commonly used.

The next hex-dump describes a file-header created with blender 2.48 on little-endian hardware with a 32 bits pointer length.

```
                              pointer-size  version-number
0000 0000: [42 4C 45 4E  44 45 52] [5F]  [76] [32 34 38]      BLEN DER_ v248
                  identifier         endianness
```

File-block

File-blocks contain a file-block-header and data. The start of a file-block is always aligned at 4 bytes. The file-block-header describes the total length of the data, the type of information stored in the file-block, the number of items of this information and the old memory pointer at the moment the data was written to disk. Depending on the pointer-size stored in the file-header, a file-block-header can be 20 or 24 bytes long. The next table describes how a file-block-header is structured.

reference	structure	type	offset	size
code	char[4]	Identifier of the file-block	0	4
size	integer	Total length of the data after the file-block-header	4	4
old memory address	void*	Memory address the structure was located when written to disk	8	pointer-size (4/8)
SDNA index	integer	Index of the SDNA structure	8+pointer-size	4
count	integer	Number of structure located in this file-block	12+pointer-size	4

by Jeroen Bakker

Code describes different types of file-blocks. The code determines with what logic the data must be read. These codes also allow fast finding of data like Library, Scenes, Object or Materials as they all have a specific code.

The size contains the total length of data after the file-block-header. After the data a new file-block starts. The last file-block in the file has code 'ENDB'.

The old memory address contains the memory address when the structure was last stored. When loading the file the structures can be placed on different memory addresses. Blender updates pointers to these structures to the new memory addresses.

```
0000 4420: [53 43 00 00]  [60 05 00 00]  [A0 2F 04 0A]  [8B 00 00 00]    SC.. `... ./.. ....
0000 4430: [01 00 00 00]  [xx xx xx xx    xx xx xx xx    xx xx xx xx     .... xxxx xxxx xxxx
```

SDNA index contains the index in the DNA structures to be used when reading this file-block-data. More information about this subject will be explained in the Reading scene information section.

Count tells how many elements of the specific SDNA structure can be found in the data.

The next section is an example of a file-block-header. The code 'SC'+0x00h identifies that it is a Scene. Size of the data is 1376 bytes (0x05h X 256 + 0x60h = 1280 + 96); the old pointer is 0x0A042FA0h and the SDNA index is 139 (8 X 16 + 11). The section contains a single scene. Before we can interpreted the data of this file-block we first have to read the DNA structures in the file. The section structure DNA will show how to do that.

Structure DNA

Structure DNA is stored in a file-block with code 'DNA1'. It can be just before the 'ENDB' file-block. It contains all internal structures of the Blender release the file was created in. The data in this file-block must be interpreted as described in this section. In a blend-file created with Blender 2.48a this section is 43468 bytes long and contains 309 structures. These structures can be described as C-structures. They can hold fields, arrays and pointers to other structures, just like a normal C-structure.

```
structure Scene {
        ID id; // 52 bytes long (ID is different a structure)
        Object *camera; // 4 bytes long (pointer to an Object structure)
        World *world; // 4 bytes long (pointer to a World structure)
        ...
        float cursor[3]; // 12 bytes long (array of 3 floats)
        ...
}
```

The next section describes how this information is ordered in the data of the 'DNA1' file-block.

repeat condition		name	type	length	description
		identifier	char[4]	4	'SDNA'
		name identifier	char[4]	4	'NAME'
		#names	integer	4	Number of names follows
for(#names)		name	char[]	?	Zero terminating string of name, also contains pointer and simple array definitions (e.g. '*vertex[3]\0')
		type identifier	char[4]	4	'TYPE' this field is aligned at 4 bytes
		#types	integer	4	Number of types follows
for(#types)		type	char[]	?	Zero terminating string of type (e.g. 'int\0')
		length identifier	char[4]	4	'TLEN' this field is aligned at 4 bytes
for(#types)		length	short	2	Length in bytes of type (e.g. 4)
		structure identifier	char[4]	4	'STRC' this field is aligned at 4 bytes
		#structures	integer	4	Number of structures follows
for(#structures)		structure type	short	2	Index in types containing the name of the structure
..		#fields	short	2	Number of fields in this structure
..	for(#field)	field type	short	2	Index in type
for end	for end	field name	short	2	Index in name

by Jeroen Bakker

As you can see, the structures are stored in 4 arrays: names, types, lengths and structures. Every structure also contains an array of fields. A field is the combination of a type and a name. From this information a catalog of all structures can be constructed. The names are stored as how a C-developer defines them. This means that the name also defines pointers and arrays. (When a name starts with '*' it is used as a pointer. when the name contains for example '[3]' it is used as a array of 3 long.) In the types, you'll find simple-types (like: 'integer', 'char', 'float'), but also complex-types like 'Scene' and 'MetaBall'. 'TLEN' part describes the length of the types. A 'char' is 1 byte, an 'integer' is 4 bytes and a 'Scene' is 1376 bytes long.

- ***Note:** While reading the DNA you'll will come across some strange names like '(*doit)()'. These are method pointers and Blender updates them to the correct methods.*
- ***Note:** The fields 'type identifier', 'length identifier' and 'structure identifier' are aligned at 4 bytes.*

The DNA structures inside a Blender 2.48 blend-file can be found at http://www.atmind.nl/blender/blender-sdna.html. If we understand the DNA part of the file it is now possible to read information from other parts file-blocks. The next section will tell us how.

Reading scene information

Let us look at the file-block we have seen earlier. The code is 'SC'+0x00h and the SDNA index is 139. The 139th structure in the DNA is a structure of type 'Scene'. The associated type ('Scene') has the length of 1376 bytes. This is exact the same length as the data in the file-block. We can map the Scene-structure on the data of the file-blocks. But before we can do that, we have to flatten the Scene-structure.

The first field in the Scene-structure is of type 'ID' with the name 'id'. Inside the list of DNA structures there is a structure defined for type 'ID' (structure index 17). The first field in this structure has type 'void' and name '*next'. Looking in the structure list there is no structure defined for type 'void'. It is a simple type and therefore the data should be read. '*next' describes a pointer. The first 4 bytes of the data can be mapped to 'id.next'. Using this method we'll map a structure to its data. If we want to read a specific field we know at what offset in the data it is located and how much space it takes.

The next table shows the output of this flattening process for some parts of the Scene-structure. Not all rows are described in the table as there is a lot of information in a Scene-structure.

reference	structure	type	name	offset	size	description
id.next	ID	void	*next	0	4	Refers to the next scene
id.prev	ID	void	*prev	4	4	Refers to the previous scene
id.newid	ID	ID	*newid	8	4	
id.lib	ID	Library	*lib	12	4	
id.name	ID	char	name[24]	16	24	'SC'+the name of the scene as displayed in Blender
id.us	ID	short	us	40	2	
id.flag	ID	short	flag	42	2	
id.icon_id	ID	int	icon_id	44	4	
id.properties	ID	IDProperty	*properties	48	4	
camera	Scene	Object	*camera	52	4	Pointer to the current camera
world	Scene	World	*world	56	4	Pointer to the current world
set	Scene	Scene	*set	60	4	Pointer to the current set
Skipped rows						
r.sfra	RenderData	int	sfra	248	4	Start frame of the scene
r.efra	RenderData	int	efra	252	4	End frame of the scene
Skipped rows						
r.xsch	RenderData	short	xsch	326	2	X-resolution of the output when rendered at 100%

by Jeroen Bakker

reference	structure	type	name	offset	size	description
r.ysch	RenderData	short	ysch	328	2	Y-resolution of the output when rendered at 100%
r.xparts	RenderData	short	xparts	330	2	Number of x-part the renderer uses
r.yparts	RenderData	short	yparts	332	2	Number of y-part the renderer uses
Skipped rows						
sculptdata.axislock	SculptData	char	axislock	1365	1	
sculptdata.pad	SculptData	char	pad[2]	1366	2	
frame_step	Scene	int	frame_step	1368	4	
pad	Scene	int	pad	1372	4	

We can now read the X and Y resolution of the Scene. The X-resolution is located on offset 326 of the file-block-data and must be read as a short. The Y-resolution is located on offset 328 and is also a short.

- ***Note:*** *An array of chars can mean 2 things. The field contains readable text or it contains an array of flags (not humanly readable).*

- ***Note:*** *A file-block containing a list refers to the DNA structure and has a count larger than 1. For example Vertexes and Faces are stored in this way.*

Next steps

The implementation of saving in Blender is easy, but loading is difficult. When implementing loading and saving blend-files in a custom tool the difficulty is the opposite. In a custom tool loading a blend-file is easy, and saving a blend-file is difficult. If you want to save blend-files I suggest to start with understanding the the global file structure and parsing the DNA section of the file. After this is done it should be easy to read information from existing blend files like scene data, materials and meshes. When you feel familiar with this you can start creating blend-libraries using the internal Blender structures of a specific release. If you don't want to dive into the Blender source code you can find them all at http://www.atmind.nl/blender/blender-sdna.html.

There is a feature request on supporting an XML based import/export system in Blender. I don't support the request, but it is interesting to look at how this can be implemented. An XML export can be implemented with low effort as an XSD can be used as DNA structures and the data can be written into XML [see http://www.atmind.nl/blender/blender-file.zip to download JAVA example including source code]. Implementing an XML import system uses a lot of memory and CPU. If you really want to implement it, I expect that the easiest way is to convert the XML-file back to a normal blend-file and then load it using the current implementation. One real drawback is that parsing a XML based blend-file uses a lot of memory and CPU and the files can become very large.

At this moment I'm using this information in an automated render pipeline. The render pipeline is build around a web-server and SVN. When an artist commits a new blend-file in SVN, it is picked up by the web-server and it will extract resolutions, frames scenes and libraries from the blend-file. This information is matched with the other files in SVN and the blend-file will be placed in the render pipeline ■

Jeroen Bakker

Jeroen (Amsterdam, the Netherlands, 33 years old) worked as coder in the demo scene. He is interested in open source and 3d animations. At the moment he is working on products supporting impact analysis and change management around a fully automated render pipeline.

Website: http://www.atmind.nl/blender
Email: j.bakker@atmind.nl

by Troy James Sobotka

Deep within the bowels of Blender, hidden behind the facade of a 3D modeler, animation system, game engine, and other arcane components, are two other aspects often overlooked. The Sequencer and Compositor are perhaps two of the most underrated and often invisible aspects of the wonderful tool known as Blender.

The following process assumes that you have an intermediate knowledge of Blender. It focuses primarily on the work-flow of motion picture production as opposed to the technical details of keystrokes and menu selections. It is also aimed primarily at working on the first half of motion picture work - the editing. Finishing and effecting could cover an entire article unto itself, and as such, it is largely glossed over in this piece.

This article was created as a result of interest generated from a music video project. (http://troy-sobotka.blogspot.com/2009/02/right-where-it-belongs.html) The project's author was rather shocked by the interest in it and the spin-off coverage (http://www.soulpancake.com/.....)

Creating Motion Pictures with Blender

Step One - Think about It

Before you go out and start shooting - think. What do you want to explore or say? What is the tone of your piece? Think about your audience. Think about why you are creating it.

Work within your inherent limitations. Clearly defining the 'playing field' of your creative project allows you to push your creativity to the limits and helps to keep focus.

Step Two - Plan and Shoot

If you are working on a project that features a score, Blender offers several useful tools to help get a preliminary pacing and previse into place. Using the timeline, you can drop markers on the fly (m) and label them (CTRL-m). Once you have your markers in place, you can experiment using sketched storyboards or like material as temporary placeholders. Import your image and drag / stretch it to meet the markers you have placed. This should help you get a rough idea of your project before you go out and shoot it.

Once you have equipped yourself with as much information as you feel is needed, go out and shoot. Experiment with certain aspects while delivering the content you feel you need according to your previse thinking.

by Troy James Sobotka

Step Three - Get the Content onto Your Computer

This isn't nearly as much of an obstacle as it has been in the past. Many consumer level digital cameras / portable video cameras deliver the material in a Quicktime wrapper and encoded using a codec that is supported by ffmpeg. To make things easier, locate your material under a single project directory. If you have many assets, feel free to subdivide the material into meaningful categories. Remember, this is your work-flow and should reflect how you work, not the way a piece of software will force you to work.

Import all of your strips into the timeline and locate it somewhere in the strips view that won't impact your project. If you have many assets, locate them to the left of the time 0 marker. This allows you to keep all of your assets loaded into Blender while keeping them out of the way of your working area. Use the name field on the clips to give them meaningful titles. When importing your assets, running Blender in a windowed mode is useful to flip between your file manager to preview the clips and Blender to import them.

If you are shooting in HD, make certain to proxy your work for a speedier and more responsive environment. To turn proxying on, simply select render size that is less than 100%. For each clip you import, select it and click the 'Use Proxy' button. Rebuilding the proxy will allow you to render a series of JPGs under your assets for each strip. From this point onward, Blender will use the appropriate proxy sequence for each preview size you render.

In the end, warp the work-flow process to meet your needs.

Step Four - Assemble the Rough Cut

If you used the marker system above, you are now free to drag and drop your clips into their respective slots and evaluate their impact. Don't stress about small details - the point of the assembly is to get all of your key moments into place. The assembly is not the place to be slipping frames and tidying cuts. This is very much a "forest through the trees" phase where the net sum goal is to have a complete visualization of the project for evaluation. Try to limit your cutting to the soft cut tool (SHIFT-k), as this will permit you to retain all of the original sequence and adjust the in and out points further along the production pipeline.

Once you have your project in a completely rough cut format, stop for a moment and get a coffee or take a break. Walk away.

When you return, evaluate where you are with the project. Do you need to go out and shoot a key element? Is the project flowing along in a manner that is working toward the goal you established at Step One? Is there something that needs reworking from where you originally visualized it in your head?

Using the rendering panels, render out a lower resolution version for analysis.

Once you have a clear idea of the strengths and weaknesses for the entire project, make notes. How can you make a given area stronger? How can you fix a problem in another area? Focus on the project as a whole and fight to not get too obsessed with the minor details in a particular sequence.

Step Five - Iterate and Tighten

Keeping an eye on the whole project is critical at this point. Your goal here should be to tighten your rough cut down and iterate over your changes. If an area is relatively stable, take this second pass as an opportunity to tighten beat points and tighten your cuts.

If you find yourself needing to refer to imported footage, use your notes and scrub through the strips by clicking and dragging on the given strips over in the logging / library area you have established.

by Troy James Sobotka

This allows you to see only the given strip isolated in the preview window.

Step Six - Commit Yourself

At some point, make the decision to put your paintbrush down. The more projects you attempt, the easier this point will be to assess and feel comfortable about. To help aid in this evaluation, ask yourself if you have touched on the goals you established in Step One.

Consider this a critical moment - this is the point where you will no longer make changes. From this moment onward, it is strictly polishing and tightening.

Step Seven - Tweak and Twiddle

This phase commences the march towards final cut. Slip your frames left and right and clean up the project. If you are timing to a score or a particular sound effect, does the cut point feel tight? Evaluate your transitions as well. Does a fade to black work better than a dissolve? Test the duration of your fades and dissolves.

Step Eight - Take a Break

Fresh eyes are critical at this point. Maybe put the project away for an hour, a couple of hours, or possibly even a sleep. When you return, you may find greater clarity for the hard and gritty analysis of your editorial decisions.

Step Nine - Final Cut

This is a final pass that is very much similar to Step Seven. At this juncture you should be doing very little other than the odd frame shifting / tweaking where required. You have agreed at this point that you will be doing no further tweaking on the editing of your project.

Step Ten - Finish

If you have any effects slated for your project, this is where you would commence work on them. Blender's compositor is extremely powerful for complex effect sequences, and as such, no brief summary would do it justice here.

You may find that you do not need the compositor if the sequencer effects meet your needs. Remember, the goal is to produce creative work and in the end, the process should be completely insignificant as you work toward that goal. If the sequencer's tools meet the needs of your project, then so be it.

Experiment with different looks and evaluate according to the goals you established in Step One. If you find something interesting but are struggling to make it work with your current project, simply call it project 'next' and make a mental note. There will always be time for another project...

In Closing

Motion picture work has now been dominated by commercial and proprietary tools. Often, there is a perception that you cannot create motion picture work without them. This piece is a challenge to every reader to put down the excuses and procrastination, pick up Blender, and prove that entire misconception wrong 🟥

Troy James Sobotka

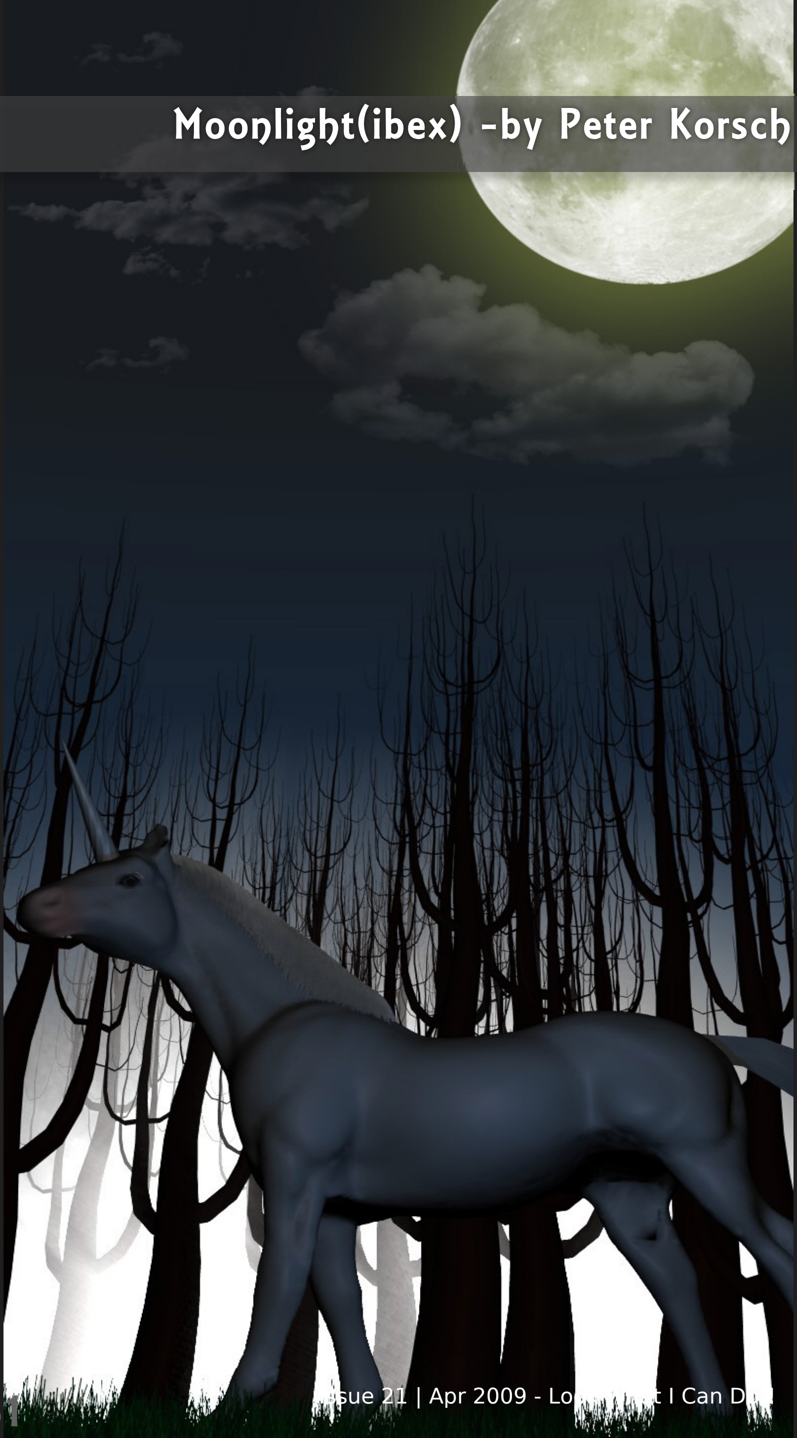
Troy James Sobotka started cutting film back when you actually had to cut celluloid with a blade and tape the two strips together. He comes from an era where the non linear editor was in its infancy. He wonders what it would have been to have a Free Software tool like Blender back then.

Website: http://troy-

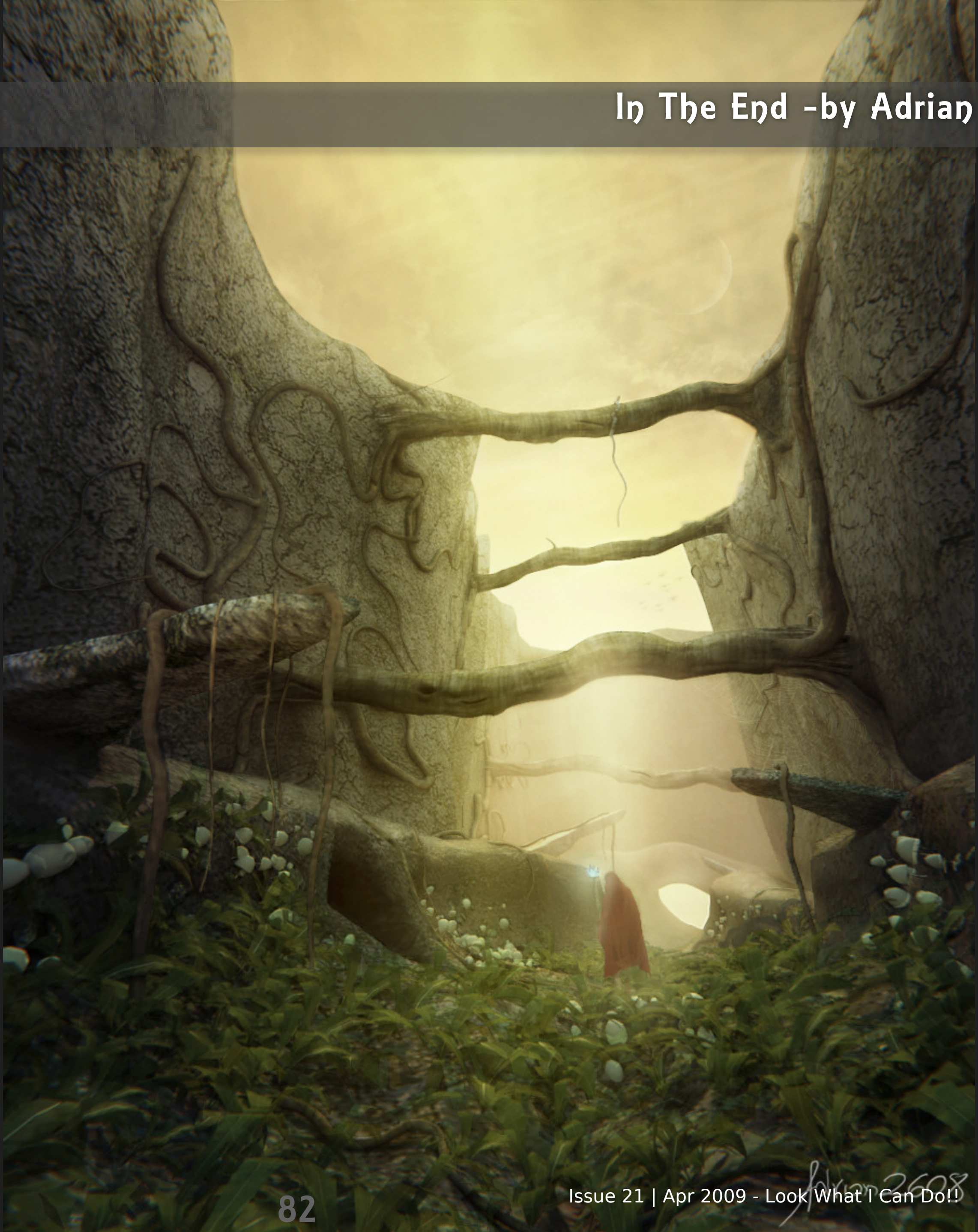
The Rake - by Olaf Schoener

Steampunk Lion -by Kevin Hays(Jeepster)

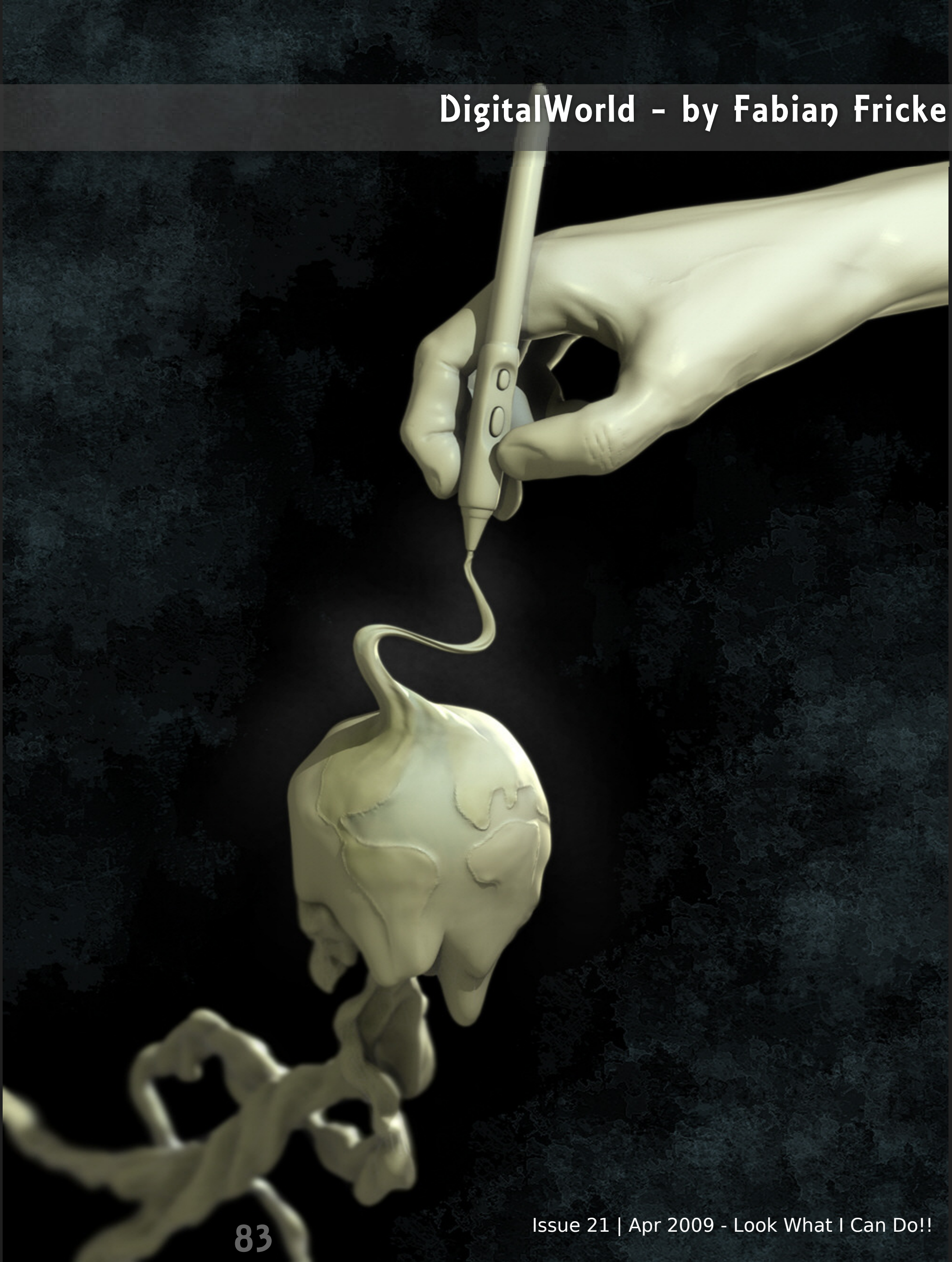
Moonlight(ibex) -by Peter Korsch



In The End -by Adrian



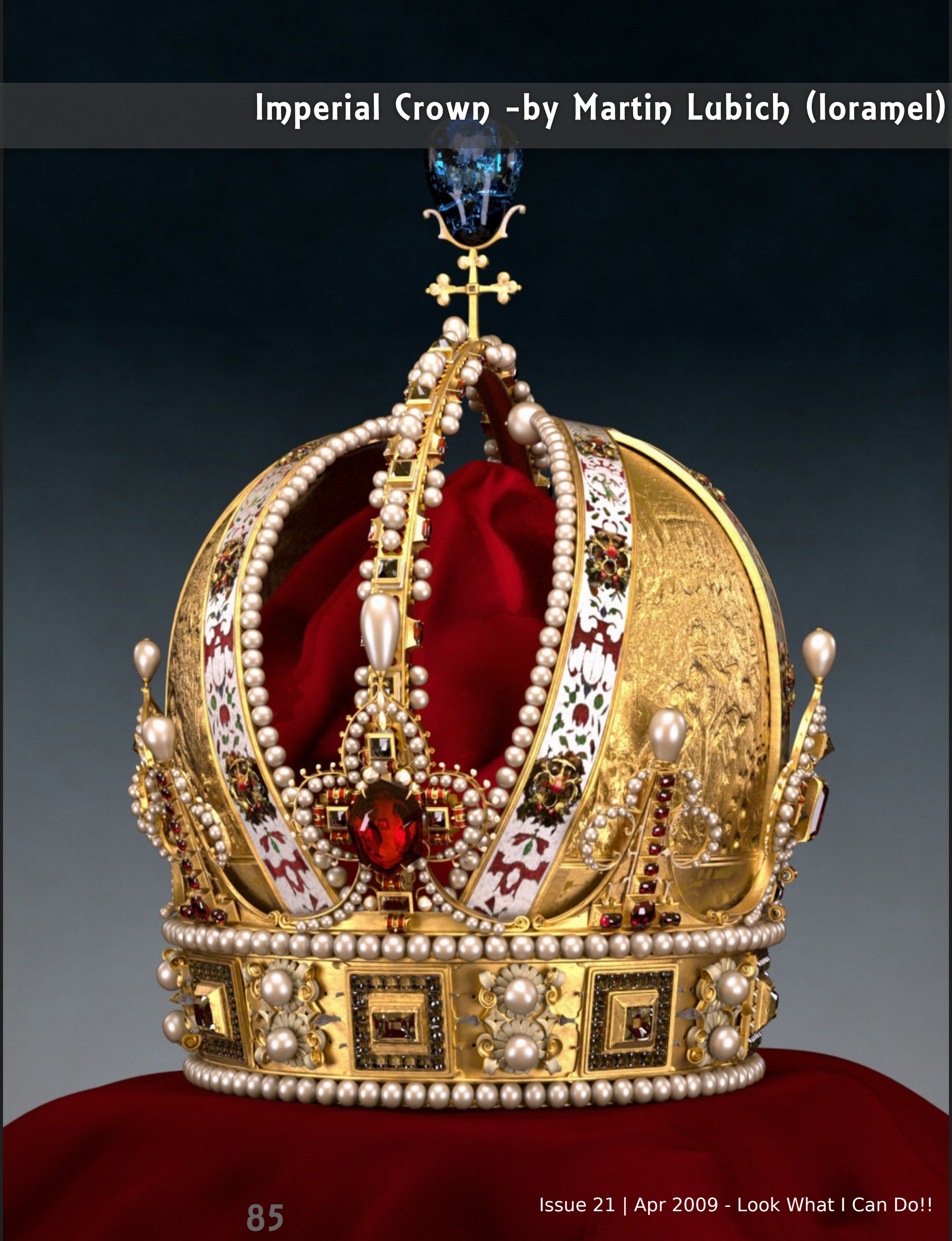
DigitalWorld - by Fabian Fricke



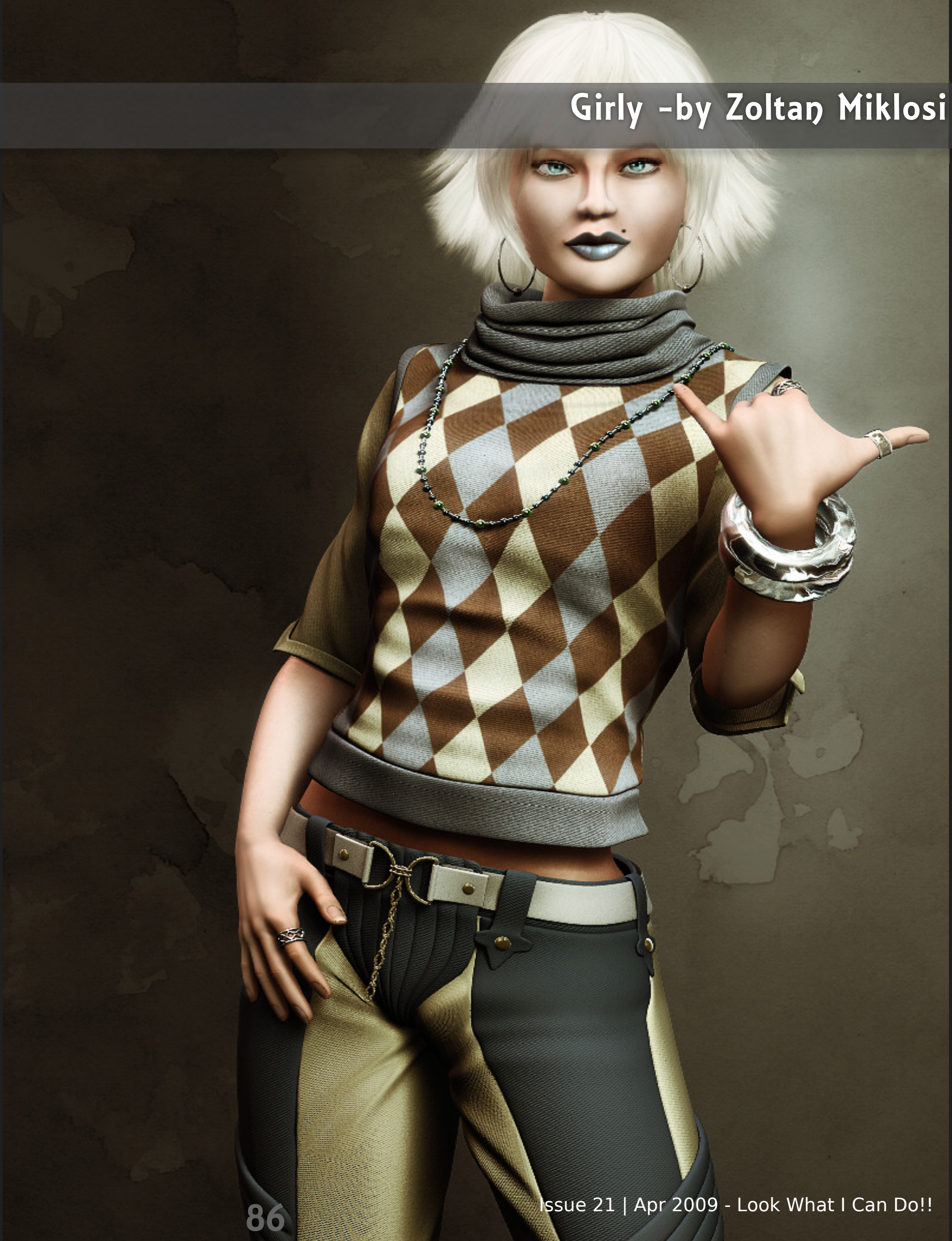
Cocktail -by Thomas Kristof

thomislav86 - blender 2.47 - indigo 1.1.9
MOJITO

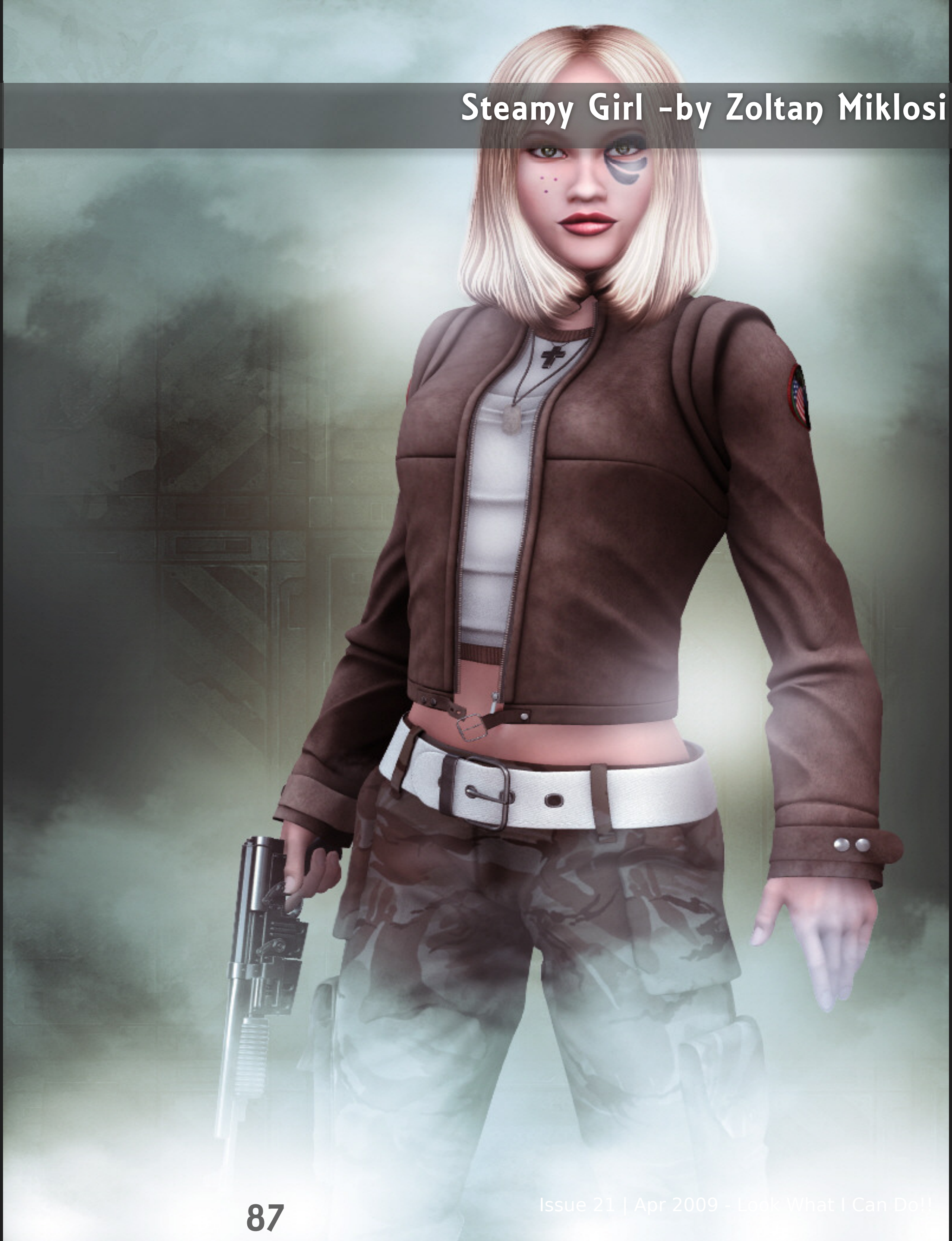
Imperial Crown -by Martin Lubich (loramel)



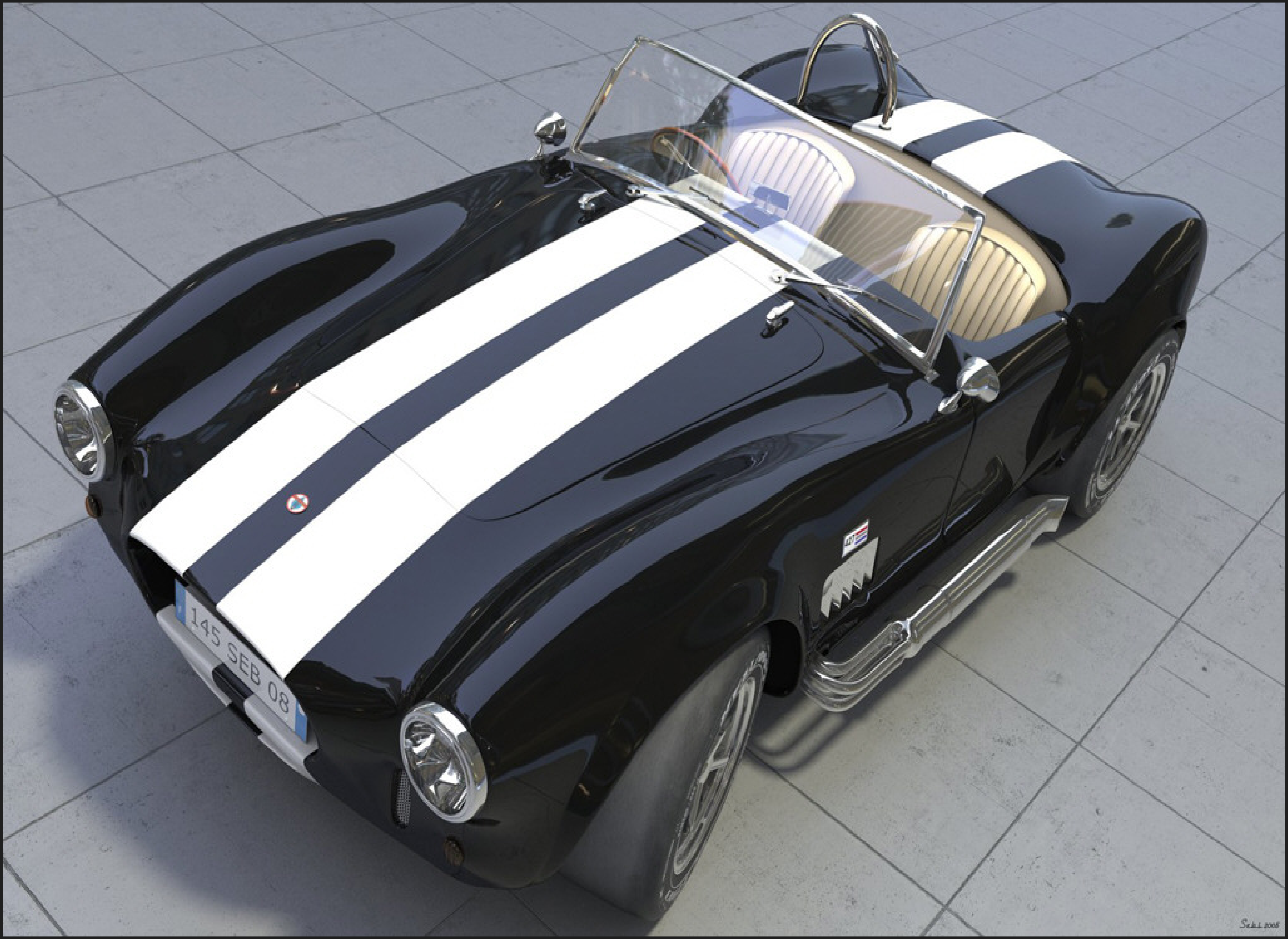
Girly -by Zoltan Miklosi



Steamy Girl -by Zoltan Miklosi



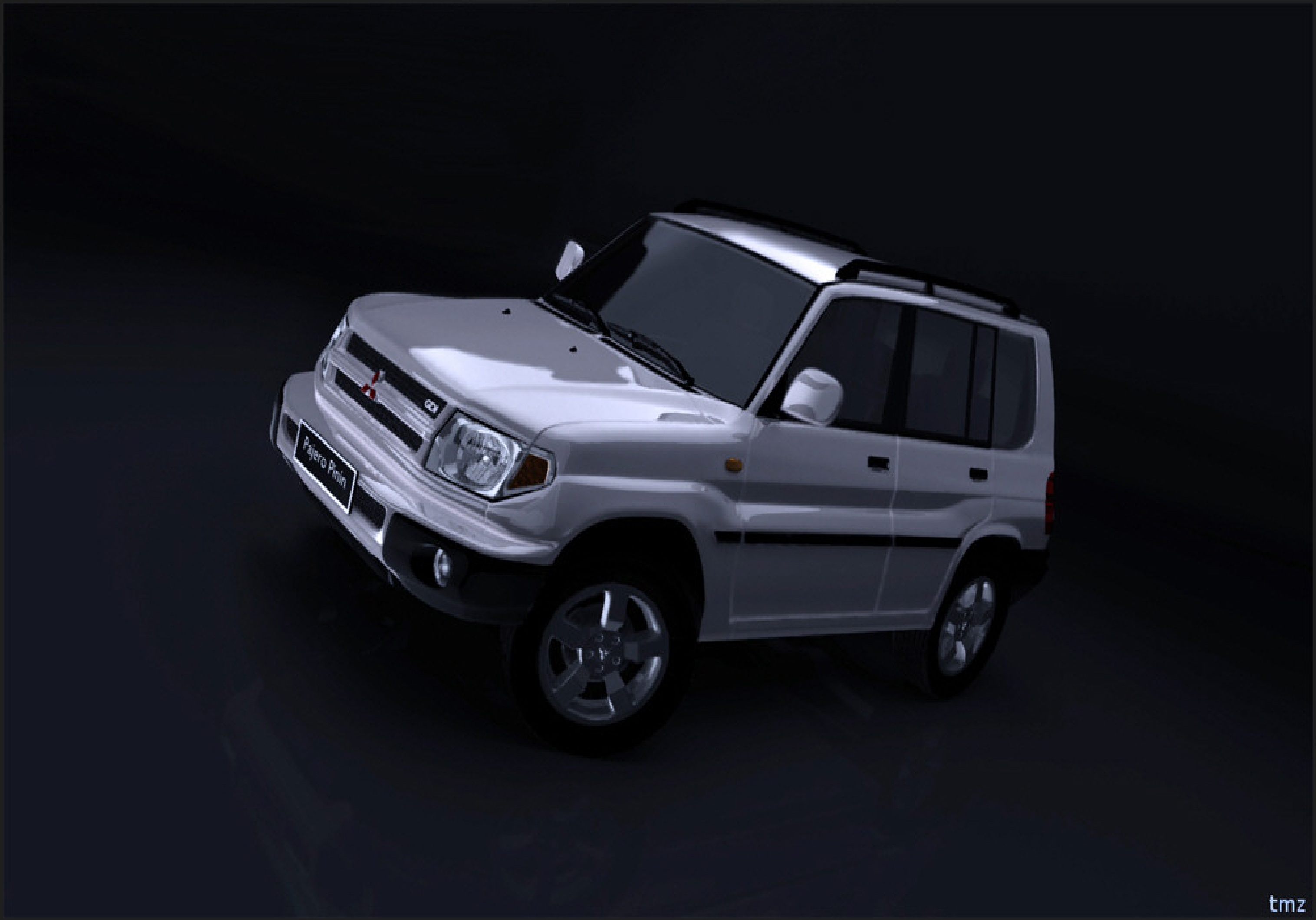
Cobra1 -by Sebastian Tribot



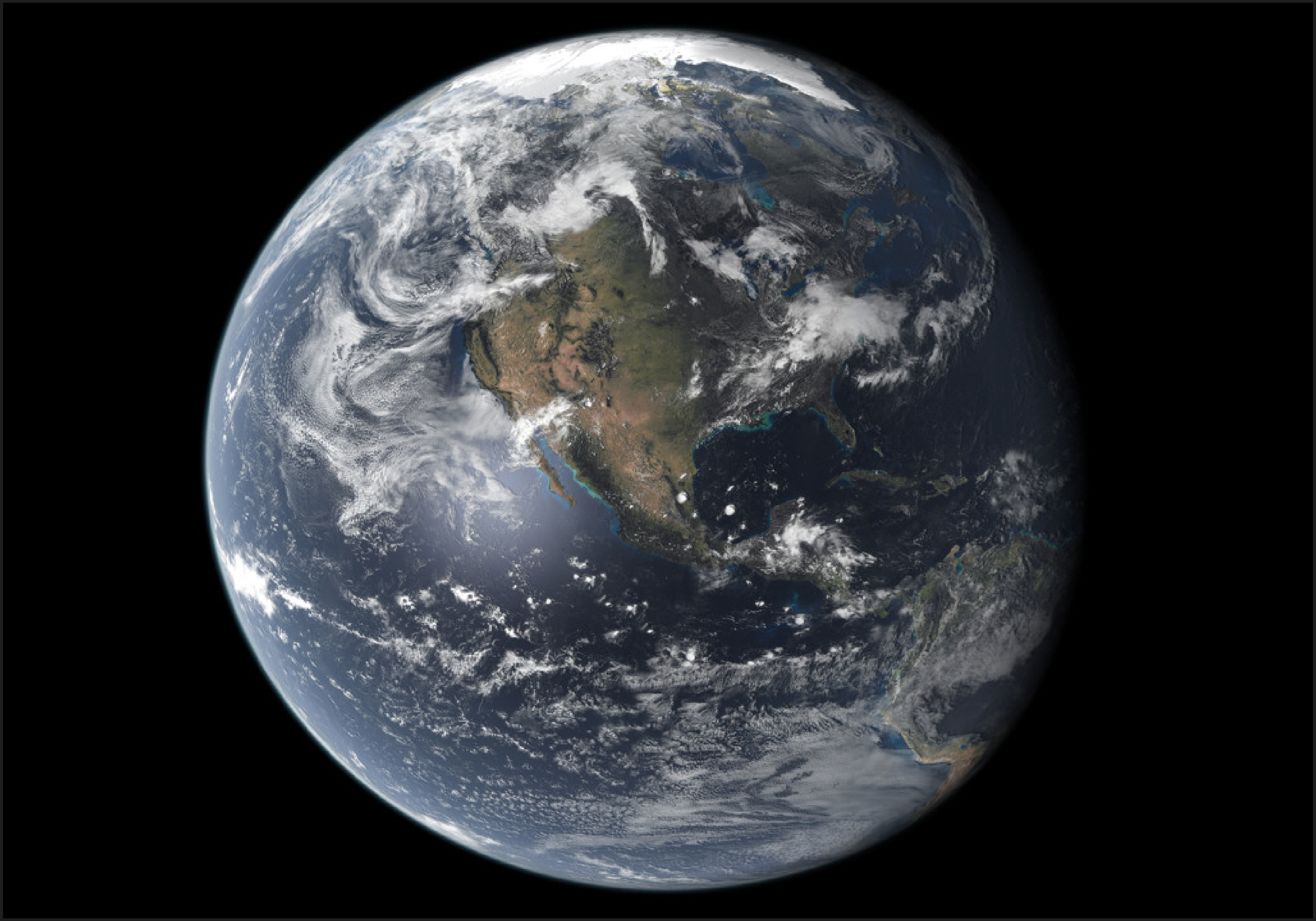
Cobra2 -by Sebastian Tribot

Sebi 2008
145 SEB 08
GOODYEAR
EAGLE

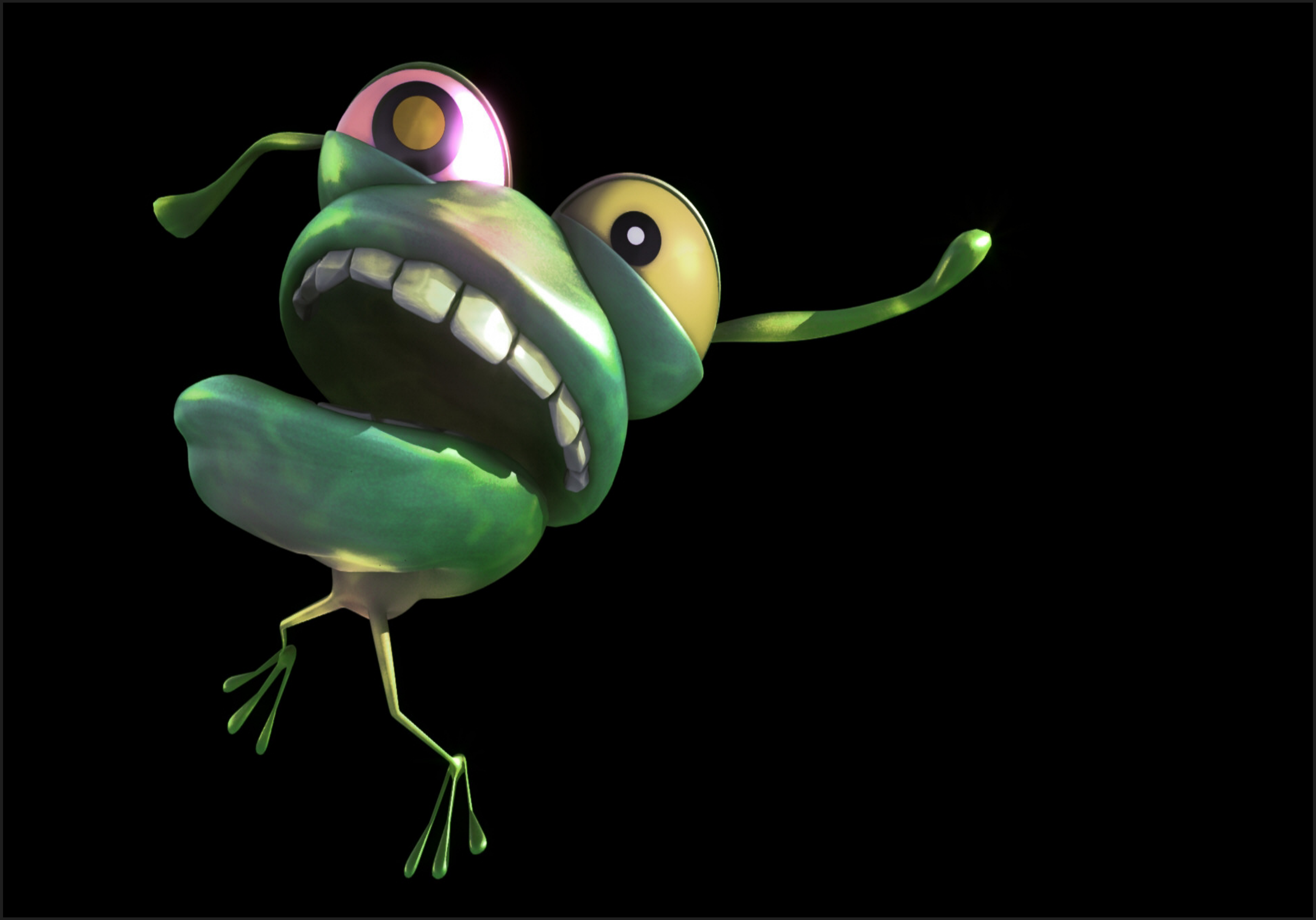
Pajero -by Antony Matzakos



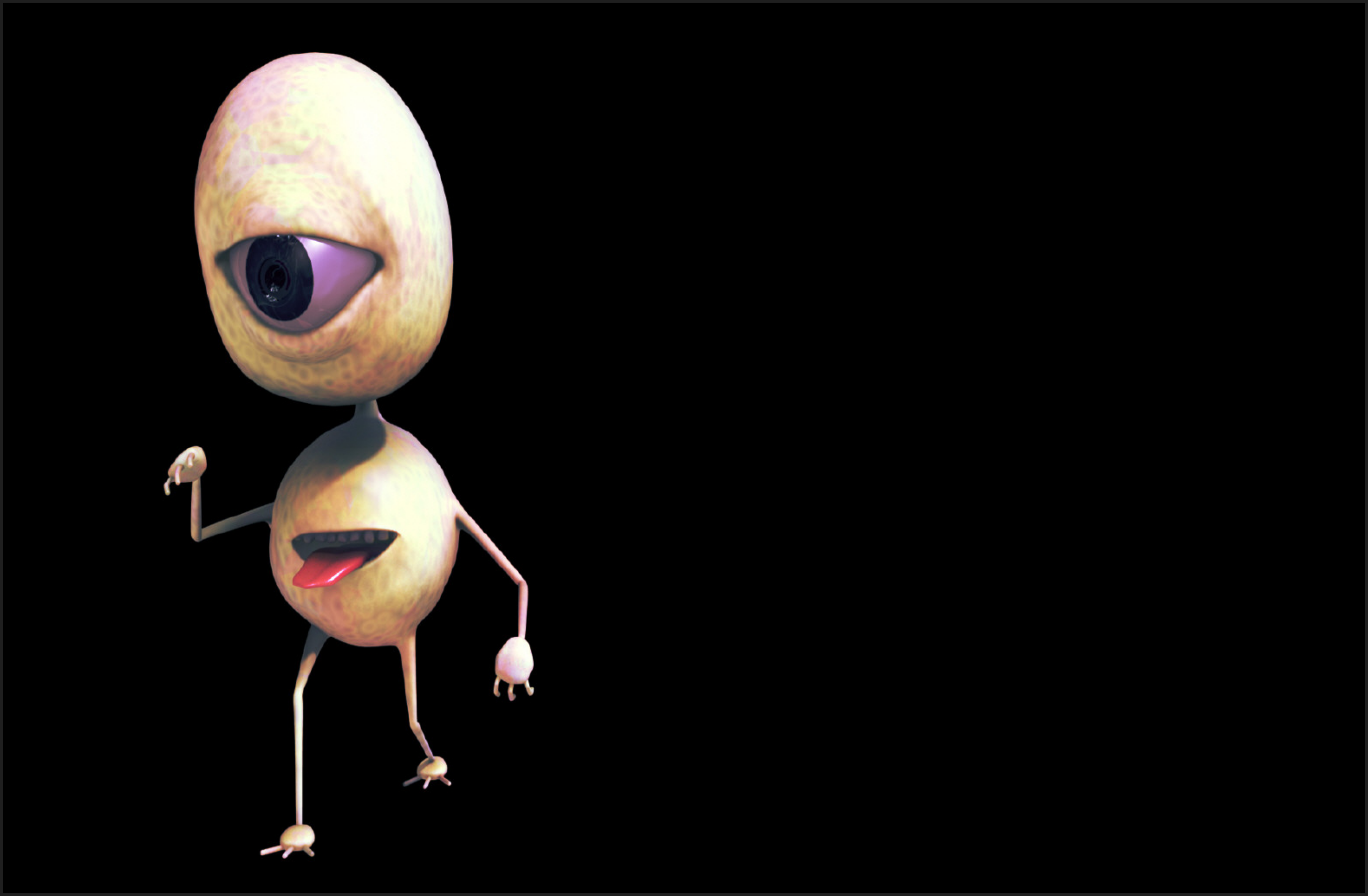
Earth -by William Chamberlin(Lego Animator)



Lambe Suex -by Budi Purnomo



Moto Melex -by Budi Purnomo



Want to write for BlenderArt Magazine?

Here is how!

1. We accept the following:

- Tutorials explaining new Blender features, 3dconcepts, techniques or articles based on current theme of the magazine.
- Reports on useful Blender events throughout the world.
- Cartoons related to blender world.

2. Send submissions to sandra@blenderart.org. Send us a notification on what you want to write and we can follow up from there. (Some guidelines you must follow)

- Images are preferred in PNG but good quality JPG can also do. Images should be separate from the text document.
- Make sure that screenshots are clear and readable and the renders should be at least 800px, but not more than 1600px at maximum.
- Sequential naming of images like, image 001.png... etc.
- Text should be in either ODT, DOC, TXT or HTML.
- Archive them using 7zip or RAR or less preferably zip.

3. Please include the following in your email:

- Name: This can be your full name or blenderartist avtar.
- Photograph: As PNG and maximum width of 256Px. (Only if submitting the article for the first time)
- About yourself: Max 25 words .
- Website: (optional)

Note: All the approved submissions can be placed in the final issue or subsequent issue if deemed fit. All submissions will be cropped/modified if necessary. For more details see the blenderart website.

Issue 22

"Things that go bump in the Night"

- Lighting for spooky, eery or night scenes.
- Characters that are spooky, creepy or scary.
- Images, Animations, Games that evoke a sense of dread/spookiness/eery caution.

Disclaimer